uc3m | Universidad Carlos III de Madrid

Grado Universitario en Ingeniería Informática
2018-2019

Trabajo Fin de Grado

"Definición, desarrollo e implantación de una plataforma empotrada HW/SW para el desarrollo de nanosatélites"

Adán Cano Moreno

Tutor/es
Javier López Gómez
Leganés, 2019

"*Aprendí que el coraje no era la ausencia de miedo, sino el triunfo sobre él. El valiente no es el que no siente miedo, sino el que vence ese temor.*" - Nelson Mandela

Agradecimientos

Este proyecto no habría sido posible sin el apoyo de mucha gente. Quisiera dedicar el proyecto a Jose Pedro Cano Duque, Covadonga Moreno Simón, Rebeca Cano Moreno y Patricia Sánchez Rocha.

Además, quisiera agradecer a toda mi familia, amigos y compañeros todo el apoyo e interés mostrado durante este tiempo. Por último, quisiera agradecer a Javier López Gómez todo el esfuerzo que ha dedicado en ayudarme a desarrollar el proyecto y por todos los consejos que me ha dado durante todo este tiempo. También agradecer a Javier Fernández Muñoz y Jesús Carretero Pérez la ayuda prestada durante el proyecto y a SENER la oportunidad de colaborar en un proyecto espacial.

Resumen

La tecnología espacial está en continua mejora. En los últimos años ha surgido la filosofía *New Space* que se basa en la colonización del espacio a través de pequeños satélites, que son más baratos y rápidos de fabricar en comparación a los satélites clásicos. Cada vez son más los servicios que son prestados al mundo que necesitan una infraestructura espacial. Además, cada vez son más las empresas que colaboran con universidades para el desarrollo de satélites para el estudio de fenómenos espaciales y el análisis del comportamiento de las nuevas tecnologías en entornos hostiles.

Este proyecto pertenece a la cátedra UC3M-SENER, una colaboración entre la Universidad Carlos III de Madrid y una empresa española del sector de la ingeniería, SENER. El objetivo de esta cátedra multidisciplinar es que estudiantes de diferentes departamentos construyan un nanosatélite con la colaboración de SENER. Este proyecto se encarga de la definición del entorno de desarrollo del software y de la integración del software del ordenador de a bordo en un sistema empotrado.

El desarrollo del sistema empotrado es fundamental, pues será el encargado de ejecutar el software de a bordo. En el proyecto el sistema empotrado se desarrolla en una BeagleBoard-XM de ARM. El software de a bordo es ejecutado sobre un sistema operativo de tiempo real RTEMS.

Para verificar el sistema, se realiza la fase de verificación *software-in-the-loop* (que prueba el sistema sobre un emulador hardware) y la fase *hardware-in-the-loop* (que prueba el sistema en el hardware real).

Palabras clave: nanosatélite, sistema empotrado, sistema operativo de tiempo real, *software in the loop* y *hardware in the loop*.

Abstract

Space technology is constantly improving. In recent years, the New Space philosophy has emerged, which is based on the colonization of space through small satellites, which are cheaper and faster to manufacture with respect to classic satellites. Lately, most of the services which are being provided to the world need a space infrastructure. In addition, some companies are collaborating with universities to develop satellites and analyze the behaviour of new technologies in hostile environments.

This project belongs to the UC3M-SENER chair, a collaboration between the Carlos III University of Madrid and the Spanish engineering company SENER. The aim of this multidisciplinary chair is for students from different departments to build a nanosatellite in collaboration with SENER. This project is responsible for the definition of the software development environment and the integration of the onboard computer software into an embedded system.

The development of the embedded system is fundamental, as it will be in charge of executing the onboard software. For this project, the embedded system is based on an ARM BeagleBoard-XM. The onboard software is executed on a Real Time Operating System.

In regards to system verification, we perform software-in-the-loop validation (which tests the system on a hardware emulator), and hardware-in-the-loop validation (which tests the system on the real hardware).

Keywords: nanosatellite, embedded system, Real Time Operating System, software in the loop and hardware in the loop.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1

Introducción

Este proyecto se encuentra enmarcado dentro de una cátedra conjunta entre la universidad UC3M y la empresa SENER. SENER es una empresa española del ámbito de la ingeniería que trabaja en sectores como el de las energías y el aeroespacial. Esta cátedra multidisciplinar, que reúne estudiantes de ingeniería informática, aeroespacial…, tiene como objetivo la construcción de un nanosatélite con la colaboración de SENER, cuya finalidad será, previsiblemente, la toma de fotografías de alta resolución de la Tierra y el estudio de comportamiento de algunos materiales al ser radiados. Se estima que la fecha del lanzamiento del nanosatélite se produzca aproximadamente en el año 2022.

Un satélite es un objeto artificial que se encuentra orbitando alrededor de planetas, lunas o asteroides. El lanzamiento del satélite hacia el espacio se realiza a través de un cohete, el cohete y la carga, en este caso el satélite, se separan, acercándose el satélite a la localización esperada en el espacio. Una vez allí, el satélite comienza a orbitar debido a diversas teorías de la física. Que el satélite orbite o no depende de su velocidad, de la gravedad que ejerce y de la inercia. La gravedad es la fuerza de atracción que ejerce una masa de un objecto sobre otro objeto. Cuando mayor sea el valor de la masa, mayor fuerza de gravedad se producirá. La inercia es la fuerza que mantiene al satélite en un movimiento recto y que los empuja hacia el espacio. Por lo tanto es gracias a la fuerza de gravedad que ejerce un satélite por lo que se mantiene en órbita. La gravedad ralentiza al satélite y lo ajusta con respecto a la curvatura del planeta.

Por otro lado, para que exista un equilibrio entre las fuerzas de gravedad e inercia. La fuerza que se debe ejercer sobre el conjunto debe ser la justa para que el satélite llegue a la localización necesaria para que comience a orbitar. La velocidad a la que orbita el satélite debe ser muy precisa, ya que si se desplaza demasiado rápido puede salirse de su órbita, lo que haría que el satélite se perdiera en el espacio. Si el satélite se mueve demasiado despacio, el satélite comenzaría a caer en dirección a la superficie sobre la que orbita.

Existen muchos tipos de órbitas que se pueden agrupar en diferentes conjuntos. Si nos fijamos en los tipos de órbitas de acuerdo a su altitud tendremos cuatro tipos de órbitas principales:(1) La Órbita Geoestacionaria (GEO) tiene una altitud de unos 36.000 kilómetros respecto a la Tierra y usa la órbita ecuatorial por lo que su periodo es de 24 horas. Esto implica que se mueve a la misma velocidad que la Tierra por lo que el satélite siempre se encontrará en la misma posición respecto a la de un observador en la Tierra.(2) La Órbita Baja (LEO) es una órbita centrada en la Tierra con una altitud entre 200 y 2.000 kilómetros y tiene un periodo orbital de unos 128 minutos o menos.(3) La Órbita Media (MEO) es una órbita geocéntrica que se encuentra entre unos 2.000 y 36.000 kilómetros respecto a la Tierra. Su periodo orbital se

encuentra entre las 2 y 24 horas.(4) La Órbita Altamente Elíptica (HEO) es una órbita elíptica que se encuentra a más de 36.000 kilómetros donde su periodo es superior a las 24 horas.

1.1. Historia

En 1957 la Unión Soviética lanzó al espacio con éxito el primer satélite en orbitar respecto a la Tierra. Este satélite recibió el nombre de Sputnik 1 y pertenecía a una familia de satélites del programa espacial soviético Sputnik [2]. Fue lanzado en Octubre de 1957 y su misión era conseguir orbitar. La elección del nombre viene del significado de la palabra en ruso, que no es otra que satélite. La idea de este programa era estudiar la viabilidad de orbitar satélites alrededor de la Tierra. A finales de la década de 1950 y a principios de la década de 1960 se lanzaron múltiples satélites Sputnik cada uno con una misión específica. A continuación se van a describir brevemente alguna de las misiones más importantes del programa.

Sputnik 2 fue el primer satélite en llevar un pasajero vivo a bordo. Era un perro con nombre Laika. La misión no contemplaba un retorno del satélite por lo que Laika murió tras un sobrecalentamiento del vehículo. El satélite fue lanzado en Noviembre de 1957. Sputnik 3 fue la primera misión del programa en fallar. Tenía el objetivo de realizar mediciones de radiación sobre el cinturón de Van Allen [1]. La grabadora falló lo que hizo de la misión un fracaso. El lanzamiento se produjo en Mayo de 1958. Sputnik 4 fue el primer satélite en incluir un maniquí en la cabina. Este tenía la función de simular y experimentar con futuros lanzamientos donde el satélite fuera controlado por una persona [2]. El lanzamiento de este satélite se realizó en Mayo de 1960. Por último, Sputnik 10 fue el último satélite del programa en ser lanzado. Tuvo lugar en Marzo de 1961. Tras múltiples lanzamientos donde se incluían maniquís y nuevas especies de animales, la misión del Sputnik 10 consistía en el lanzamiento de un satélite que contenía un maniquí, un perro, un sistema de televisión y diversos aparatos científicos para experimentar sus funcionalidades.

Después del Sputnik 1, se han producido múltiples lanzamientos de satélites. Según la *Oficina de las Naciones Unidas para Asuntos del Espacio Extraterrestre (UNOOSA)*, se han lanzado alrededor de 8000 objetos artificiales donde se incluyen satélites, cohetes, etc. A lo largo del tiempo, con la mejora de la tecnología se ha podido incluir nuevos sistemas como el uso de sistemas empotrados para el control del satélite.

1.2. Sistemas Empotrados

Un sistema empotrado es un controlador que es manejado por un RTOS cuyo diseño se basa en realizar unas tareas específicas con restricciones de tiempo real. Un RTOS (*Real Time Operating System*) es un sistema operativo que ha sido desarrollado para aplicaciones de tiempo real , y que debe cumplir unos requisitos temporales para que el funcionamiento sea el adecuado [3]. La mayoría de los componentes de un sistema empotrado se encuentran en la placa base como por ejemplo la tarjeta gráfica o el módem. La mayoría de los sistemas empotrados modernos son microcontroladores que permiten que el sistema vea reducido su tamaño. Un microcontrolador es un circuito integrado programable, capaz de ejecutar las órdenes almacenadas en su memoria. Los diseñadores se encuentran en continuo estudio sobre la posibilidad de optimizar el sistema

[1] Zona del espacio cercana a la Tierra que contiene partículas cargadas energéticamente

[2] Estos lanzamientos se agruparían dentro del programa soviético Vostkov

para poder reducir el tamaño y el coste y mejorando el rendimiento de la realización de las tareas. Actualmente la mayoría de los sistemas empotrados se producen en masa para aumentar los beneficios.

Los principales fabricantes de sistemas empotrados son Intel, Motorola, AMD y STMicroelectronics. Alguno de ellos son capaces de integrar el microprocesador y los elementos de control de los elementos de entrada y salida en un único chip para cumplir con las especificaciones de los sistemas que han sido descritos anteriormente. Esta técnica es aplicada a sistemas que no requieren mucha potencia ya que la capacidad de proceso es inferior a la de los sistemas empotrados tradicionales.

En cuanto a los sistemas operativos que son necesarios para que un sistema empotrado pueda funcionar y ejecutar programas suelen ser sistemas especiales para sistemas empotrados. Son sistemas que tienen pocos requisitos de memoria, que incluyen la posibilidad de contener aplicaciones en tiempo real y que permiten incorporar únicamente los módulos necesarios para un sistema empotrado en concreto. Los más conocidos en la actualidad son FreeRTOS, RTEMS y VxWorks.

Como se ha mencionado anteriormente, un ejemplo de la constante mejora en la tecnología relacionada con los satélites son los sistemas empotrados para el control de los mismos. Esto también se puede aplicar a un *CubeSat*[4]. Un *CubeSat* es un satélite de menor tamaño cuyo peso máximo no sobrepasa los 1.3 kilogramos. Su tamaño estándar es de 10x10x10 centímetros lo que hace que tenga la forma de un cubo. Los *CubeSats* pueden combinarse para formar un nanosatélite. Un nanosatélite es cualquier satélite que pese menos de 10 kilogramos. El uso de los nanosatélites va en aumento, ya que pueden ejecutar las mismas funciones que un satélite tradicional, lo que provoca que cada vez haya más objetos artificiales en el espacio.

Hoy en dia, los nanosatélites se pueden usar para localizar, con un alto porcentaje de éxito, cualquier objeto independientemente de su posición en la Tierra. Algunos ejemplos de localización de objetos realizada por un nanosatélite son:

Sistema ADS-B : El sistema de vigilancia de transmisión automática permite conocer la localización de un avión sin tener que hacer uso de las señales de radar. La información que envía el nanosatélite es usada para garantizar el control y la seguridad de los pasajeros así como de la mercancía.

Control de barcos : La comunicación que se produce entre el nanosatélite y los barcos permite que se pueda averiguar la posición de un barco en el mar, para poder controlar la situación en todo momento y verificar el estado de la mercancía. Además, se realiza el seguimiento de los barcos para aumentar su seguridad, gestionar las rutas marítimas, facilitar el rescate en caso de accidente, etc. Por otro lado, que se realice un seguimiento de los barcos permite que se reduzca la piratería así como la pesca ilegal y otras acciones que dañan el medioambiente.

Gestión de flotas : Este ejemplo suele ser aplicado por empresas que tienen flotas para controlar la mercancía que transportan o, en el caso de empresas de transporte de civiles, un control de la situación en tiempo real del vehículo (empresas como *Uber* o taxis tradicionales). Los datos que son enviados a las empresas pueden ser usado para mejorar la productividad de la compañía.

Hoy en día, está aumentando el uso de los nanosatélite para realizar experimentos con la nueva tecnología que va a apareciendo como es el caso de los transistores de grafeno. El sistema

de a bordo de un nanosatélite necesita tener una estructura específica similar a la de la Figura 1.1. En esta figura podemos ver como las aplicaciones y servicios del nanosatélite se ejecutan sobre el sistema operativo, independientemente de qué sistema operativo se use.

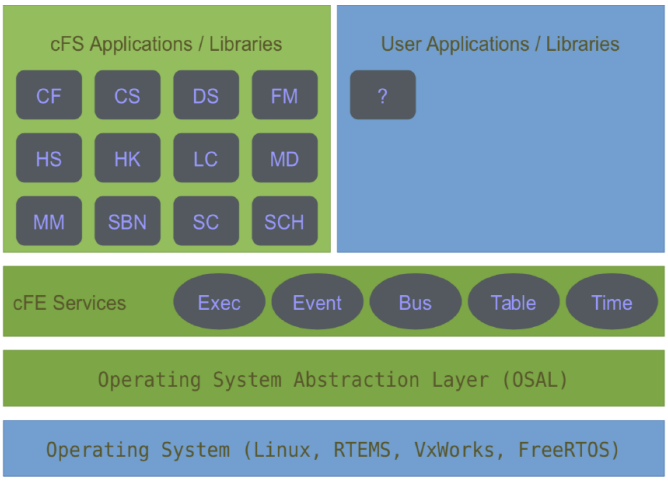


Figura 1.1: Ejemplo de la estructura del On-Board Software [1]

1.3. Motivación

La arquitectura de los nanosatélites surge debido a la aparición de la filosofía *"New space"*. Esta es una corriente que ha cambiado el panorama espacial tal y como lo conocíamos. Tiene como objetivo la colonización del espacio a través de pequeños satélites que son más baratos y rápidos de fabricar en comparación a los satélites clásicos. Esta corriente ha permitido que empresas que actualmente no poseen satélites en el espacio, y estiman que es necesario tener una infraestructura en el espacio para poder prestar sus servicios hayan podido entrar en el negocio espacial. Sobre todo, estas empresas son *"startups"*, empresas de nueva creación y que para entrar en un mercado competitivo necesitan tener dicha infraestructura en el espacio.

Como los nanosatélites son capaces de prestar el mismo servicio que un satélite y son menos costosos, el número de empresas que aplican esta filosofía está aumentando. Esto provoca que la tecnología que aplican los nanosatélites es cada vez más usada en el panorama espacial, por lo que este proyecto utiliza tecnología que en la actualidad está en alza.

1.4. Objetivos

A continuación se procede a enumerar los objetivos que se pretende alcanzar en el desarrollo de este proyecto. En alguno de los objetivos se incluyen objetivos secundarios que tienen relevancia en el desarrollo.

Integrar un sistema empotrado que incorpore un RTOS . Se debe generar una imagen de disco que contenga un sistema empotrado con un RTOS. Esta imagen debe ser integrada en un sistema empotrado juntos con los ficheros de configuración y arranque del sistema. La ejecución de dicha imagen debe ser probada tanto en un simulador de hardware como en hardware real. Este objetivo contiene varios objetivos secundarios necesarios para el estudio de la viabilidad del desarrollo.

Pruebas con un sistema empotrado basado en Linux. Se debe generar una imagen de disco para un sistema empotrado basado en *Linux*. Esta imagen se generará con propósito de estudio de viabilidad del sistema y contiene únicamente el conjunto de herramientas BusyBox.

Generar una imagen arrancable de un RTOS para arquitectura ARM. Se debe generar una imagen de disco que contenga un sistema empotrado de un sistema en tiempo real RTEMS compilado para la arquitectura ARM. Al ejecutar dicha máquina, se debe mostrar la ejecución de una aplicación compilada para RTEMS-ARM.

1.5. Estructura del documento

En esta sección se exponen los diferentes capítulos que componen el proyecto. Cada capítulo incluye una breve descripción para facilitar la comprensión de cada capítulo. El contenido se divide en los siguientes capítulos:

Estado del arte. Incluye información de los diversos proyectos que son similares a éste y realiza una comparativa entre ellos.

Análisis y diseño. Incluye los requisitos de usuario, de sistema, los casos de uso, la arquitectura, las matrices de trazabilidad.

Implementación. Incluye detalles respecto al desarrollo de diversos componentes del proyecto. Además, se incluye la enumeración de las diversas pruebas que verifican el funcionamiento y estabilidad del software.

Plan de proyecto. Muestra la planificación de las tareas de las que consta el proyecto e incluye un diagrama de Gantt en el que se puede observar la planificación. Además, incluye el presupuesto necesario para llevar a cabo el proyecto.

Entorno socio-económico y Marco legal. Muestra el Impacto socio-económico del proyecto así como la legislación aplicable al mismo.

Conclusiones y trabajos futuros. Incluye las conclusiones obtenidas de la realización del proyecto junto con la enumeración las mejoras y trabajos futuros que derivan del proyecto.

En este documento se incluyen apéndices donde se describen pasos específicos para desarrollar el proyecto así como algunos fragmentos de código de la implementación. En el Apéndice C se muestra un manual de instalación de las herramientas básicas necesarias para el desarrollo del proyecto. En el Apéndice A se incluye un resumen en inglés del proyecto.

Capítulo 2

Estado del arte

En este capítulo se describe, con detalle, los diferentes elementos a tener en cuenta para entender el desarrollo del proyecto. Además, se incluyen otros proyectos que tienen misiones similares al que se está exponiendo en este documento.

2.1. Sistema empotrado

Como se ha descrito en la introducción, un sistema empotrado es un controlador cuyo objetivo es realizar unas tareas específicas, típicamente con restricciones de tiempo real. Una arquitectura básica de un sistema empotrado contiene:

- **Un microprocesador**, que se encarga de ejecutar el código de las tareas y gestionar el funcionamiento global del sistema.

- **Una memoria principal**, para almacenar el código de las diferentes tareas del sistema. Esta memoria debe permitir el acceso de lectura y escritura lo más rápido posible para que el microprocesador no espere demasiado tiempo y ejecute las tareas lo más rápido posible. La memoria ha de ser no volátil para que los datos se encuentren almacenados aunque el sistema esté falto de energía. La memoria principal tiene un tamaño de entre 8 y 256 MB.

- **Una memoria caché**, que sea más rápida que la principal y que almacene únicamente aquellos datos y códigos que han sido accedidos últimamente. Esto nos permite ahorrar tiempo de acceso a los datos ya que no es necesario acceder a la memoria principal si la instrucción o el dato al que se quiere acceder se encuentra en la caché. Como la tecnología que usa este tipo de memorias es muy cara, la memoria caché tiene un tamaño de almacenamiento de entre 8 y 512 KB.

Algunos sistemas empotrados pueden omitir algún componente, e.g. memoria caché, o integrar más de uno en un solo chip, como los microcontroladores y System-on-a-Chip.

2.2. Plataformas Hardware-Software

Como se ha mencionado anteriormente, un sistema empotrado es un controlador cuyo objetivo se basa en realizar unas tareas específicas con restricciones de tiempo real.

Una plataforma, en el ámbito de la informática, hace referencia al sistema que sirve como base para que módulos hardware y software funcionen correctamente en su conjunto. Al definir una plataforma es necesario establecer el tipo de arquitectura y el sistema operativo. A continuación se van a exponer diversos ejemplos de arquitecturas que hardware. Es importante conocer las diferentes arquitecturas y procesadores para el proceso de implementación de la plataforma, puesto que el desarrollo de la plataforma empotrada dependerá de la arquitectura que estemos usando. Además, se deben conocer sus características para escoger aquella que mejor se ajuste a las necesidades del proyecto.

i386. Es un microprocesador CISC (Computador con Conjunto de Instrucciones Complejas) que pertenece a la familia de microprocesadores x86, desarrollada por Intel. Un microprocesador CISC tiene un conjunto de instrucciones que destaca por permitir operaciones complejas entre operandos situados en memoria o en los registros.

IA64. Es una arquitectura diseñada y desarrollada por Intel. Usa direcciones de memoria de 64 bits y está basada en el modelo EPIC (procesamiento de instrucciones explícitamente en paralelo) donde el compilador decide qué instrucciones ejecutar en paralelo [5]. Los procesadores que llevan esta arquitectura son conocidos como Intel Itanium. Recientemente Intel notificó que los procesadores Intel Itanium habían alcanzado su ciclo de fin de vida, por lo que no se fabricará ninguna CPU de esta familia [6].

AMD64 (X86-64). Es la versión de 64 bits del conjunto de instrucciones x86. Soporta una cantidad alta de memoria virtual y memoria física que permite que los programas puedan almacenar gran cantidad de datos en la memoria [5]. El diseño y desarrollo es obra de AMD, aunque ya ha sido implementada por empresas como Intel, entre otras.

PowerPC. Es una arquitectura de computadoras de tipo RISC (Computador con Conjunto de Instrucciones Reducidas), que fue desarrollada por *IBM, Motorola, y Apple*. RISC es utilizado en aquellos microcontroladores en los que sus instrucciones son de tamaño fijo y que son presentadas en un número limitado de formatos. Además, en las arquitecturas RISC, las instrucciones de carga y almacenamiento son las únicas que acceden a memoria [7]. Actualmente Apple ha dejado de colaborar con esta arquitectura ya que decidió que sus dispositivos comenzaran a usar las arquitecturas de Intel. Sin embargo, podemos ver ejemplos de dispositivos que usan esta arquitectura en las videoconsolas *Xbox 360 y PS3*. Otro ejemplo de esta arquitectura se encuentran en sistemas empotrados como el que lleva incluido los *Mars Rovers* de la NASA.

SPARC. Es una arquitectura con un conjunto de instrucciones reducidas (RISC)[8]. Inicialmente fue diseñada por *Sun Microsystems* [9]. Es una arquitectura RISC de acceso libre, por lo que cualquiera puede participar en el diseño de la arquitectura. La CPU SPARC está compuesta por una unidad de enteros, que procesa la ejecución básica, y una unidad de coma flotante que ejecuta las operaciones y cálculos de números reales. Ambas unidades tienen la posibilidad de estar o no estar integradas en el mismo chip.

ARM. Es la arquitectura RISC de 32 bits y 64 bits que más se utiliza en las unidades producidas. En sus inicios fue diseñada por *Acorn Computers* para su uso en ordenadores personales [8]. Los procesadores ARM requieren un menor número de transistores que los

procesadores x86 CISC, por lo que se produce una reducción de los costes y energía. Los procesadores ARM se han convertido en los dominantes dentro del mercado de la electrónica móvil e integrada, siendo microprocesadores y microcontroladores pequeños, con un bajo consumo y relativamente baratos.

En el caso de este proyecto, se ha decidido utilizar utilizar una arquitectura ARM, ya que es la arquitectura que mejor se adapta a las necesidades del proyecto. ARM tiene una amplia gama de microprocesadores que soportan un RTOS y el consumo de los mismos es relativamente bajo. Esto es importante si se tiene en cuenta que la única fuente de abastecimiento eléctrico del nanosatélite será la energía solar, y el nanosatélite no podrá completar su misión si la plataforma empotrada no tiene energía suficiente, por lo que cuanto menos consuma menor probabilidad de quedarse sin energía habrá. Entrando más en detalle, la arquitectura de la CPU de ARM utiliza técnicas de microarquitectura para soportar una amplia gama de puntos de rendimiento [8]. La arquitectura de la CPU define el conjunto de instrucciones básicas, así como los modelos de excepción y memoria en los que se basan el sistema operativo (y el hipervisor de haberlo). La microarquitectura define la potencia, el rendimiento y el área del procesador mediante la determinación de la longitud del *pipeline*, los niveles de caché, etc. Originalmente, la arquitectura de la CPU se basaba en los principios de la computadora RISC e incorporaba una arquitectura de carga y almacenamiento, en la que el procesamiento de datos funcionaba sólo con el contenido del registro, y no con el contenido de la memoria. Incluía modos de direccionamiento simples, en los que todas las direcciones de carga y almacenamiento se determinaban a partir del contenido del registro y de los campos de instrucción. En cuanto a los perfiles de arquitectura ARM, existen tres:

Arquitectura tipo A: se utiliza en áreas de aplicación informáticas complejas, como servidores y teléfonos móviles [10].

Arquitectura tipo R: se utiliza cuando se requiere respuesta en tiempo real. Por ejemplo, aplicaciones críticas para la seguridad o que necesitan una respuesta determinista, como equipos médicos o el frenado y la señalización de vehículos [11].

Arquitectura tipo M: se utiliza cuando la eficiencia energética, el consumo de energía y el tamaño son importantes. Actualmente, los dispositivos de *IoT (Internet of things)* simples se han convertido en una aplicación clave de las CPUs de perfil M. Ejemplo de uso de este perfil se pueden ver en sensores pequeños, módulos de comunicación y productos para el hogar inteligentes [12].

Los diferentes perfiles de arquitectura y números de versión se escriben como Armv8-A, Armv7-R, Armv6-M. A, R y M se refieren a los perfiles de arquitectura relevantes y 6, 7 y 8 se refieren a las diferentes versiones de la arquitectura.

2.3. Plataformas con arquitectura ARM

Como se ha mencionado anteriormente, se ha decidido desarrollar el sistema en una placa ARM. Actualmente hay una amplia gama de placas para la arquitectura ARM, puesto que tienen soporte para el RTOS escogido y el precio de estas no es muy elevado. Tras un estudio de la viabilidad de dichas placas para este proyecto, se ha optado por utilizar tipo BeagleBoard [13]. BeagleBoard es un tipo de placa ARM de hardware libre y de bajo consumo. Es producida por

Texas Instruments con la colaboración de *Newark element14* y *DigiKey*. La placa fue desarrollada por un equipo de ingenieros cuyo objetivo era el de diseñar una placa capaz de ser usada en el campo educacional en los distintos colegios del mundo, de manera que se pudiera demostrar en las clases las capacidades de un software y hardware libre. Texas Instruments usó este tipo de placas para demostrar las capacidades del *OMAP3530*. Los dispositivos OMAP3530 se basan en la arquitectura *OMAP3* [14]. La arquitectura OMAP3 está diseñada para proporcionar procesos de vídeo, imagen y gráficos suficientes para soportar vídeo en tiempo real, videoconferencia e imagen de alta resolución. Los dispositivos OMAP3530 soportan sistemas operativos de alto nivel como Linux, Windows y Android. Además, las placas ARM permiten la ejecución de sistemas en tiempo real como es el caso de RTEMS. Se pueden encontrar los siguientes tipos de placas BeagleBoard [15]:

BeagleBoard Original. Es una placa de desarrollo lanzada en octubre del 2011. Entre sus características destacan:

- Procesador AM3358 ARM Cortex-A8.
- Memoria RAM de 256 MB.
- Reloj de procesador corriendo a 720 MHz.

Figura 2.1: BeagleBoard Original

La placa BeagleBoard Original incluye conexiones JTAG con depuración de hardware, por lo tanto no es necesario un emulador JTAG. También conocida como BeagleBone. La placa BeagleBone tiene un precio de 84,99 €, según la compañía *DigiKey* [16].

BeagleBone Black. Es una plataforma de desarrollo de bajo coste para desarrolladores y aficionados. Fue lanzada al mercado en Abril de 2013. Entre sus características destacan [17]:

- Procesador AM3358 ARM Cortex-A8.
- Memoria RAM de 512 MB.
- Reloj de procesador corriendo a 1 GHz.

La placa BeagleBone Black incluye una memoria flash de 2GB. En Mayo de 2014 fue lanzada una versión, BeagleBone Black C, que aumentó el tamaño de la memoria flash a 4GB. La placa BeagleBone Black tiene un precio de 54,99 €, según la compañía *DigiKey* [18].

BeagleBone Blue. Es una placa todo en uno basado en Linux para robótica, que integra en una sola placa pequeña. Entre sus características destacan [19]:

- Procesador AM335x ARM Cortex-A8.

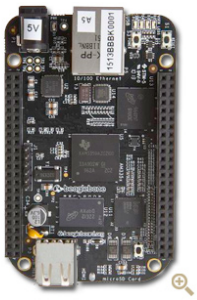


Figura 2.2: BeagleBone Black

- Memoria RAM de 512 MB.
- Reloj de procesador corriendo a 1 GHz.

Figura 2.3: BeagleBone Blue

La placa incluye una batería para autoalimentarse. Tiene un precio de mercado de 82,99 €, según la compañía *DigiKey* [20].

BeagleBoard-X15. Es la placa más sofisticada y de mayor rendimiento de la familia BeagleBoard. Entre sus características destacan [21]:

- Procesador AM5728 ARM Cortex-A15.
- Memoria RAM de 2 GB.
- Reloj de procesador corriendo a 1.5 GHz.

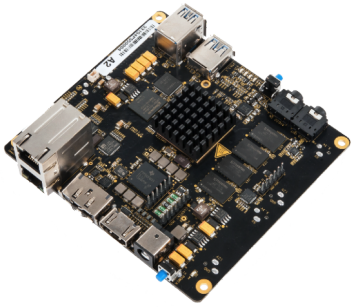


Figura 2.4: BeagleBoard-X15

BeagleBoard-X15 incluye una interfaz de alta velocidad para cada necesidad de conectividad. Su precio ronda los 215,99 €, según la página web *Arrow* [22].

BeagleBoard-XM. Es una modificación de la BeagleBoard Original. Fue lanzada al mercado en agosto de 2010. Entre sus características destacan [23]:

- Procesador AM37x ARM Cortex-A8.
- Memoria RAM de 512 MB.
- Reloj de procesador corriendo a 1GHz.

Figura 2.5: BeagleBoard-XM

BeagleBoard-xM elimina la NAND, esto hace que el sistema operativo y los datos tengan que ser almacenados en una tarjeta microSD. Su precio en el mercado ronda los 144,00 €, según la compañía *DigiKey* [24].

La Tabla 2.1 hace una comparación entre las características de las placas BeagleBoard anteriormente mencionadas.

Características	Beable-Board Original	BeableBo-ne Black	BeableBo-ne Blue	BeableBoard-X15	BeableBoard-XM
Procesador	AM335x ARM Cortex-A8	AM335x ARM Cortex-A8	AM335x ARM Cortex-A8	2x AM5728 ARM Cortex-A15	AM37x ARM Cortex-A8
Max. velocidad Procesador	720MHz	1GHz	1GHz	1.5GHz	1GHz
Memoria RAM	256MB DDR2 RAM	512MB DDR3 RAM	512MB DDR3 RAM	2GB DDR3 RAM	512 MB LPDDR RAM
Memoria flash	—	4GB 8-bit eMMC	4GB 8-bit eMMC	4GB 8-bit eMMC	—
Memoria externa	MMC/SD	microSD	microSD	microSD	MMC/SD
USB	USB JTAG	USB 2.0 480Mbps, USB 2.0	USB 2.0 480Mbps, USB 2.0	3x USB 3.0 5Gbps, USB 2.0 480Mbps	4x USB 2.0
Conexión	Ethernet	10/100 Et-hernet	WiFi, Bluetooth 4.1, BLE	2x 10/100/1000 Ethernet	4x 10/100 Ethernet
Soporta RTEMS (Board Support Packages)	Sí (beagle-boardorig)	Sí (beagle-boneblack)	—	—	Sí (beagle-boardxm)
Precio base	84,99 €	54,99 €	82,99 €	215,99 €	144,00 €

Tabla 2.1: Comparativa de las características placas BeagleBoard

Tras haber analizado las necesidades del proyecto y las diferentes placas hardware disponibles, se ha decidido usar la BealgeBoard-XM, puesto que es la que mejor se adapta los requisitos del proyecto en relación prestaciones-precio. Es una placa con soporte para un RTOS como es el caso de RTEMS, además de su bajo consumo y de sus componentes.

2.4. Sistemas Operativo en Tiempo Real

Como se ha mencionado en la introducción, un sistema operativo de tiempo real (RTOS) es un sistema operativo que ha sido desarrollado para aplicaciones de tiempo real [3]. Estos sistemas deben cumplir unos requisitos temporales para que el funcionamiento sea el adecuado [25]. Existen dos grandes grupos:

Sistemas críticos. Son aquellos que deben garantizar la realización de todas las tareas dentro de su plazo de respuesta correspondiente. Si falla un plazo, el sistema puede fallar fatalmente. También es conocido como *hard RTOS*. Ejemplos de RTOS críticos son el sistema *anti-lock breaking system* (ABS) de un vehículo o un marcapasos.

Sistemas no críticos. Son aquellos que pueden permitirse el incumplir algún plazo de respuesta de forma esporádica. Dentro de este grupo se pueden diferenciar dos tipos de RTOS: los no críticos puros (o soft RTOS, que intenta cumplir los plazos pero si no, el sistema sigue funcionando, aunque degradado) y los no críticos firmes (o firm RTOS, donde los resultados obtenidos fuera de plazo no sirven pero el sistema no se colapsa. Ejemplos de un RTOS no críticos son el procesamiento de vídeo o el sistema de climatización.

Un RTOS se caracteriza por presentar requisitos en cinco áreas principales:

Determinismo. Un RTOS es determinista siempre que realiza operaciones en tiempos fijos y predeterminados o en intervalos de tiempo predeterminados. Si en un instante de tiempo hay múltiples procesos que compiten por recursos, un sistema no es determinista. Para satisfacer las solicitudes a los recursos de manera determinista, el sistema depende varios factores: de su velocidad de la capacidad de respuesta a las peticiones y de la capacidad de gestionar todas las peticiones que le llegan. Para medir el grado de determinismo en un sistema podemos observar el retardo máximo, que consiste en el tiempo que transcurre entre que llega una interrupción con prioridad alta hasta que comienza el servicio que tiene la rutina asociada. En un RTOS este retardo es de aproximadamente de 1 milisegundo.

Sensibilidad. Este requisito hace referencia al tiempo que consume el sistema en tratar una interrupción, es decir, el tiempo que pasa desde que reconoce la interrupción hasta que da servicio a la misma.

Control del usuario. El usuario tiene el control para poder asignar prioridades a los procesos, decidir el algoritmo de planificación y qué procesos deben permanecer en todo momento en la memoria del sistema.

Fiabilidad. Un RTOS controla qué sucesos están teniendo lugar en el entorno. Las pérdidas de tiempo en el sistema que los controla pueden tener graves consecuencias.

Tolerancia a fallos. EL RTOS debe estar diseñado para responder a diferentes formas de fallos. Se pretende que el sistema pueda observar su capacidad máxima y los máximos

datos posibles cuando se produce un fallo. La idea es que el RTOS sea capaz de corregir el problema o minimizar sus efectos antes de continuar con la ejecución.

Debido a la disponibilidad de diversos RTOS, en los siguientes párrafos se van a describir, brevemente, alguno de ellos para mostrar la diferencia entre los mismos.

VxWorks. Es un RTOS desarrollado por *Wind River Systems* [26]. Suele ser usado en sistemas empotrados que requieren una respuesta rápida de alrededor de milisegundos ante las interrupciones que le llegan. Su estructura se basa en un micro-kernel de un tamaño aproximado de 20KB, un entorno de trabajo y un conjuntos de aplicaciones. Normalmente, se trabaja con él a través de una conexión remota. Un ejemplo del uso de VxWorks se encuentra en los robots *Rover* destinados con fines geológicos por la NASA [27]. El RTOS controlaba la unidad de transporte del *Rover* y proporcionaba las comunicaciones a la tierra.

FreeRTOS. Es un RTOS para dispositivos empotrados cuya licencia se distribuye bajo la licencia MIT [28]. FreeRTOS está diseñado para ser pequeño y simple. El kernel consiste en sólo tres archivos escritos en lenguaje C, para que el código sea legible, fácil de transportar y mantener. A pesar de eso, incluye algunas funciones escritas en lenguaje ensamblador como por ejemplo las incluidas en las rutinas de planificación de arquitectura. FreeRTOS no permite la gestión avanzada de memoria. También es compatible con las bibliotecas de la capa de transporte y la de sockets más populares, como *wolfSSL*.

MarteOS. Es un RTOS para aplicaciones integradas que sigue el subconjunto mínimo *POSIX.13* en tiempo real [29]. El sistema es desarrollado por el Grupo de Computación y Tiempo Real de la Universidad de Cantabria. MarteOS proporciona un entorno fácil de usar y controlado para desarrollar aplicaciones multi-hilo *Real-Time*. La mayor parte del código está escrito en Ada, aunque hay partes escritas en C y ensamblador. Todos los servicios tienen una respuesta limitada en el tiempo.

RTEMS. Es un RTOS desarrollado como software libre y diseñado para sistemas empotrados que requieren una respuesta rápida, una cierta seguridad y estabilidad [30]. Admite interfaces de programación de aplicaciones estándar como *POSIX*. Su uso se puede encontrar en vuelos espaciales, medicina, redes, etc. RTEMS es el RTOS seleccionado para el desarrollo del proyecto.

La estructura de RTEMS se basa en micro-kernel, un entorno de desarrollo y varias aplicaciones diseñadas para hacer funcionar el dispositivo correctamente. Normalmente, se trabaja con él a través de una conexión remota, donde se desarrolla el software que va a ser cargado en el dispositivo final. Entre otras características destacan:

- Está disponible para arquitecturas como *ARM, PowerPC, Intel o SPARC*.
- RTEMS tiene una estructura de organización de los datos basado en el estándar definido por *POSIX*, donde los sistemas de ficheros soportados incluyen: un sistema de archivos de tipo tabla de asignación de archivos (FAT), un sistema de datos propio (*RTEMS File System*), y el protocolo de red NFS.
- Es compatible con los protocolos de internet UDP y TCP.
- El núcleo tiene capacidad multitarea con localización de memoria dinámica.

- Es compatible con protocolos de comunicación USB, tarjetas SD/MMC, entre otros.

En el proyecto se ha decidido usar RTEMS debido a que sus características se adaptan a las necesidades del proyecto, ofrece respuesta rápida y estabilidad. Además, RTEMS se encuentra disponible para la mayoría de sistemas empotrados y es un software de acceso libre. RTEMS tiene soporte completo para el *framework* de desarrollo cFE. El procedimiento para instalar RTEMS en una máquina virtual se puede observar en el Apéndice C.

2.5. cFE

Core Flight Executive (cFE)- véase Figura 1.1- es un software desarrollado por la NASA (*National Aeronautics and Space Administration*) de acceso libre que sirve como entorno para desarrollar aplicaciones y ejecutarlas [31]. cFE provee diversos servicios que incluyen software de mensajería, tratamientos de eventos, etc. cFE define una interfaz de programación de aplicaciones (API) para cada servicio que sirve de base para el desarrollo de aplicaciones.

El servicio de software de mensajería proporciona un sistema de mensajería de publicación y suscripción que permite a las aplicaciones conectarse y comunicarse entre ellas fácilmente en el sistema. Estas aplicaciones se suscriben a los diferentes servicios en tiempo de ejecución, lo que permite que se puedan realizar modificaciones en el sistema de una manera más sencilla. Además, cFE permite que las nuevas aplicaciones puedan compilarse, cargarse e iniciarse sin tener que reconstruir todo el sistema. Para permitir la reutilización de un proyecto, cFE contiene un conjunto configurable de requisitos y código. Estos parámetros configurables permiten que el cFE se adapte a cada entorno. Esto permite que se pueda probar la aplicación en un dispositivo y, tras comprobar que funciona correctamente, desplegar la aplicación en el sistema empotrado sin necesidad de tener que modificar el software.

El cFE forma parte del *Core Flight System (CFS)* [32], una plataforma y un software reutilizable que es independiente del proyecto y que contiene un conjunto de aplicaciones reutilizables. Existen tres características principales en la arquitectura del *CFS*: es un entorno dinámico en tiempo de ejecución, es un software por capas y su diseño está basado en componentes. Debido a estas tres características y al estar la implementación del mismo dirigida a un software empotrado podemos, reutilizar el software en cualquier otro proyecto de vuelo.

Como se pudo observar en la Figura 1.1, las capas del sistema incluyen una capa de abstracción del sistema operativo (*OSAL*), una capa de ejecución del vuelo (cFE) y una capa de aplicación (donde se encuentran las diferentes aplicaciones del nanosatélite). La capa del cFE es ejecutada sobre la capa *OSAL*. *OSAL* está disponible a través de código libre y una vez se integra en el entorno de construcción cFE, se puede construir y ejecutar el sistema, y comenzar a desarrollar las diferentes aplicaciones necesarias para el proyecto.

Por lo tanto, el principal objetivo de cFE es generar la base para una plataforma y un marco software reautilizable. La combinación del cFE con el *OSAL* permite el desarrollo de software en un sistema empotrado que es independiente del RTOS y de la plataforma de hardware.

2.6. U-Boot

Como el proyecto se basa en integrar una imagen de disco en un sistema empotrado, ésta debe tener un sistema de arranque que permita inicializar el hardware para ejecutar el sistema empotrado. Para ello, se debe usar un sistema de arranque menos pesado de el *GRUB* de Linux. En este proyecto se ha decidido usar *U-Boot* ya que este sistema de arranque se encuentra en la mayoría de sistemas empotrados y permite empaquetar las instrucciones del kernel para arrancar el sistema operativo del dispositivo. *U-Boot* es un sistema de software libre que soporta varios tipos de arquitecturas, entre ellas ARM, MIPS y x86 [33].

La ROM o BIOS del sistema carga U-Boot desde un dispositivo de arranque compatible, como por ejemplo una tarjeta SD. El arranque por medio de U-Boot puede ser dividido en dos etapas [34]: la plataforma carga un SPL (*Secondary Program Loader*), que realiza la configuración inicial del hardware y carga la versión completa de U-Boot. A diferencia de otros sistemas de arranque que seleccionan automáticamente las ubicaciones de memoria del núcleo y otros datos de arranque, U-Boot requiere que se especifiquen las direcciones de memoria como destinos para copiar los datos.

U-Boot no necesita leer un sistema de ficheros para que el núcleo lo use como sistema de ficheros raíz o disco RAM inicial. Proporciona un parámetro específico al núcleo, y copia los datos a la memoria. Sin embargo, U-Boot también puede leer un sistemas de ficheros. De esta manera, en lugar de que los datos que U-Boot carga se almacenen en una ubicación fija en el dispositivo de almacenamiento, U-Boot puede leer el sistema de archivos para buscar y cargar el núcleo, el árbol de dispositivos, etc, a través de la ruta. Entre los sistemas de ficheros que soporta U-Boot se encuentran FAT 32 y ext4.

2.7. Proyectos similares

Con la aparición de la filosofía *"New Space"*, el desarrollo del proyecto es más rápido y menos costoso por lo que son cada vez más las empresas que dedican sus esfuerzos en construir infraestructuras en el espacio para prestar sus servicios. Según la consultora *Frost and Sullivan* se prevén unos 11.500 lanzamientos de nanosatélites hasta el año 2030[35]. Por lo tanto, estamos frente a un entorno altamente competitivo. Esta rapidez de producción y bajo coste permite que universidades de todo el mundo puedan desarrollar nanosatélites.

Tras realizar una búsqueda exhaustiva de diferentes misiones de nanosatélites como la que se está desarrollando en este proyecto, se ha descubierto infinidad de misiones similares. Se ha analizado en profundidad misiones desarrolladas por universidades españolas de prestigio nacional e internacional, como son la Universidad Politécnica de Madrid y la Universidad Politécnica de Cataluña. Además se han buscado proyectos internacionales, donde destacaban las colaboraciones entre universidades y la NASA, pero no se ha encontrado la información suficiente para realizar un análisis detallado y poder compararlos con los demás. Por otro lado, se ha decidido incluir el primer nanosatélite desarrollado por un grupo español, el Xatcobeo. En las siguientes secciones se describen alguno de ellos.

CubeCat-1

El 29 de Noviembre de 2018 fue lanzado el nanosatélite *CubeCat-1* desde la base india de *Sriharikota*. CubeCat-1[36] es un nanosatélite satélite de la familia de satélites CubeCat de la Universidad Politécnica de Cataluña (UPC). Una de las misiones que tiene que realizar este nanosatélite es experimentar como se comportan los transistores de grafeno en condiciones extremas.

Otra de las misiones a realizar es el estudio del efecto de las partículas energéticas altamente cargadas a través de un contador Geiger [1]. Además, en el proyecto se incluye una cámara fotográfica para tomar fotografías de la Tierra desde el espacio.

Características

- El sistema de a bordo contiene una placa con la CPU *ATM91SAM9G20*(ARM9 core) a 400 MHz con 64 MB de SDRAM.
- El software de vuelo está compuesto por un conjunto de procesadores, librería y drivers.
- El sistema operativo es un kernel Linux sobre el que es ejecutado el hipervisor en tiempo real *Xenomai*.
- El software implementa funcionalidades como la gestión de energía, el control del estado del sistema, el planificador de tareas, la ejecución de las pruebas experimentales, el protocolo de comunicación y un sensor de monitorización.

CubeCat-2

El 15 de agosto de 2016 fue lanzado el nanosatélite *CubeCat-2* desde la base china de *Jiuquan*[37]. CubeCat-2[38] es un nanosatélite desarrollado por la UPC. Una de las misiones que tiene que realizar este nanosatélite es experimentar con el uso de señales de navegación para la observación de la Tierra. La tecnología usada para las señales de navegación es el *Sistema de Posicionamiento Global* (GPS). Además, este proyecto tiene el objetivo de realizar experimentos como cargar *payloads* para experimentos científicos así como para analizar el comportamiento de materiales.

Características

- El sistema de a bordo contiene un microcontrolador ARM7 a 40 MHz.
- El software de vuelo contiene un sistema operativo de un solo núcleo.
- Sobre el sistema operativo es ejecutado el sistema en tiempo real *FreeRTOS*.
- El software implementa funcionalidades como la lectura de sensores, el protocolo de comunicación, el manejo de comandos y la gestión de los datos y los payloads.

[1] instrumentos para medir partículas radiactivas

CubeCat-3

CubeCat-3[39] es un nanosatélite que actualmente está siendo desarrollado por la UPC. Se pretende que el lanzamiento del CubeCat-3 se produzca en el año 2020. Al ser un proyecto en desarrollo, se desconocen muchas de las características del mismo. Actualmente se está analizando la viabilidad de una misión con una carga útil de imágenes multiespectrales.

Características

- El sistema de a bordo contiene un microcontrolador ARM Cortex A5 a 500 MHz.
- El software de vuelo se ejecutará sobre Linux usando el hipervisor Xenomai.

CubeCat-4

El CubeCat-4 es el cuarto miembro de la familia de nanosatélites del NanoSat Lab de la UPC [40]. Se pretende que el lanzamiento del CubeCat-4 se produzca en el año 2020. La misión de este proyecto tiene por objetivo demostrar las capacidades de los nanosatélites para la observación de la Tierra utilizando el Sistema Mundial de Navegación por Satélite - Reflectometría (GNSS-R) y la radiometría de microondas en banda L, así como para los Servicios de Identificación Automática.

Características

- El sistema de a bordo contiene un microcontrolador ATM91SAM7A1 ARM7 a 40 MHz.
- El software de vuelo contiene un sistema operativo que contiene el sistema de tiempo real *FreeRTOS*.

QBITO

QBito es el primer nanosatélite desarrollado por la Universidad Politécnica de Madrid (UPM)[41]. El nanosatélite forma parte del proyecto *QB50*, un proyecto internacional liderado por el instituto belga Von Karman, que tiene la misión de estudiar las características de la baja atmósfera. El nanosatélite fue lanzado en 18 de Abril de 2017 a unos 400 kilómetros de altura[42], donde es imposible llegar con globos aerostáticos ni con satélites, ya que para los primeros son alturas demasiado altas, mientras que para los segundos son demasiado bajas. Es por esto por lo que no se han podido realizar un mayor análisis de la capa de la baja atmósfera anteriormente. Es en la baja atmósfera donde se absorbe gran parte de la radiación de la energía que llega desde el Sol, que es la más dañina para el planeta. La duración de la misión se estimó de alrededor de seis meses.

Características

- Contiene un detector de infrarrojo de onda media sin necesidad de refrigeración.
- El sistema de a bordo contiene un microcontrolador ARM Cortex-M3 que corre a 40 MHz.

- El software del ordenador a bordo es desarrollado en el lenguaje C y está basado en el sistema operativo en tiempo real *FreeRTOS*.

Xatcobeo

Xatcobeo[43] es el primer nanosatélite desarrollado por un equipo español. Fue desarrollado por la universidad gallega de Vigo e INTA. Llevaba a bordo tres elementos principales: una radio reconfigurable a través de software, un sistema de medición de niveles de radiación ionizante y un sistema de despliegue de paneles solares. El objetivo de la misión era realizar experimentos en el ámbito de las comunicaciones y en el uso de la energía fotovoltaica en los satélites. La energía fotovoltaica es la transformación directa de la radiación solar en electricidad. Un uso que permitiría a los satélites poder abastecerse de electricidad a través de la radiación solar. Fue lanzado en Febrero del 2012 y estuvo funcionando dos años y medio cuando su vida útil estimada era de tres meses[44].

Características

- El sistema de a bordo contiene una placa Virtex-II FPGA.

2.8. Comparativa de los proyectos

El punto principal en común entre todos los proyectos, incluyendo este, es que se trata de proyectos en los que los desarrolladores son estudiantes de universidad. El principal objetivo, además de lanzar el nanosatélite, es que los estudiantes puedan colaborar en un proyecto espacial a través de la colaboración con equipos experimentados. Como se ha podido apreciar en la descripción de cada uno de los proyectos, las misiones eran muy similares. Los nanosatélites eran lanzados al espacio para realizar estudios, ya sea del espacio o de la Tierra, y probar dispositivos tecnológicos en un entorno tan hostil como es el espacio. Como se ha indicado en el Capítulo 1, el desarrollo del nanosatélite tiene la misión de captar fotografías de la Tierra y experimentar con aparatos tecnológicos. Por lo tanto, este proyecto no se diferencia a los otros proyectos en cuanto a su misión general.

A pesar de ser todos los proyectos bastante similares, cada uno tiene su importancia ya que no realizan los mismos estudios ni los mismos experimentos sobre la tecnología. En la Tabla 2.2 se resume las características de los diferentes proyectos analizados. En la Tabla 2.3 se muestran las misiones de cada uno de los proyectos.

Características	CubeCat-1	CubeCat-2	CubeCat-3	CubeCat-4	QBITO	Xatcobeo	Este proyecto
CPU	ARM9	ARM7	ARM Cortex A5	ARM 7	ARM Cortex-M3	Virtex-II FPGA	—
Velocidad del reloj	400 MHz	40 MHz	500 MHz	40MHz	40 MHz	—	—
Sistema en tiempo real	Xenomai	FreeRTOS	Xenomai	FreeRTOS	FreeRTOS	—	RTEMS
Desarrollado por	UPC	UPC	UPC	UPC	UPM	UVIGO	UC3M
Peso	1.2 kg.	7.1 kg	10 kg	1.18 kg	2 kg	1 kg	—
Inversión privada	✓	✓	✓	✓	✓	✓	✓
Open source	✓	✓	✓	✓	✓	✓	✓
Fecha lanzamiento	2018	2016	—	—	2017	2012	—

Tabla 2.2: Comparativa de las mis de los proyectos analizados

Proyecto	Misión del proyecto
CubeCat-1	Estudiar el efecto de las partículas energéticas altamente cargadas a través de un contador Geiger y experimentar como se comportan los transistores de grafeno en condiciones extremas.
CubeCat-2	Experimentar con el uso de señales de navegación para la observación de la Tierra. La tecnología usada para las señales de navegación es el *Sistema de Posicionamiento Global* (GPS).
CubeCat-3	Se está analizando la viabilidad de una misión con una carga útil de imágenes multiespectrales.
CubeCat-4	demostrar las capacidades de los nanosatélites para la observación de la Tierra utilizando el Sistema Mundial de Navegación por Satélite - Reflectometría y la radiometría de microondas en banda L, así como para los Servicios de Identificación Automática.
QBITO	Estudiar las características de la baja atmósfera.
Xatcobeo	Realizar experimentos en el ámbito de las comunicaciones y en el uso de la energía fotovoltaica en los satélites.
Este proyecto	Tomar fotografías de alta resolución de la Tierra y estudiar el comportamiento de algunos materiales al ser radiados.

Tabla 2.3: Comparativa de las misiones de los proyectos analizados

Capítulo 3

Análisis y diseño

En este capítulo se va a realizar un estudio del proyecto, tanto para definir sus funcionalidades como sus características. Para ello, se incluye una visión general del proyecto así como los requisitos del usuario con las características del mismo. De los requisitos de usuario derivan los requisitos del sistema. Además, en este capítulo se incluye la matriz de trazabilidad de los requisitos especificados.

Los requisitos de usuario han sido obtenidos por el grupo de estudiantes de la cátedra del departamento de informática, actuando como analistas durante las reuniones celebradas con los profesores de la cátedra y los trabajadores de SENER.

En este proyecto, se ha decidido seguir la metodología de desarrollo en V, tal y como se puede observar en la Figura 3.1.

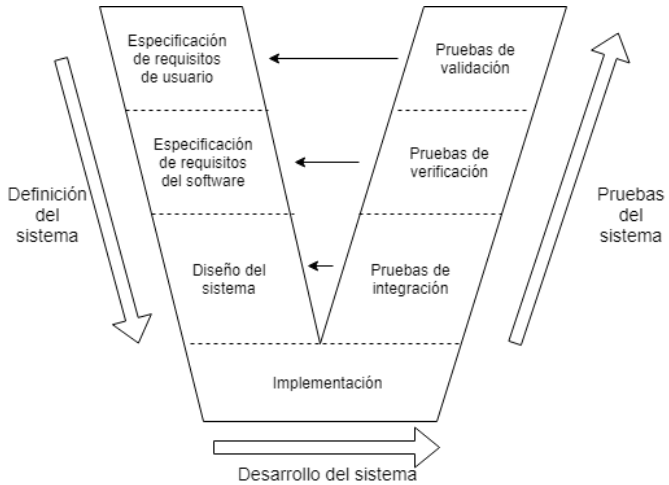


Figura 3.1: Esquema de método de desarrollo en V (la imagen ha sido cedida por Pablo Andreu Sedeño, jefe de proyecto)

Como se puede apreciar en la figura anterior, lo primero que se hace es definir los requisitos de usuario junto con el cliente y a su vez se definen las pruebas de validación que se encargarán de verificar que el producto cumple las necesidades del cliente. A continuación, se especifican los requisitos de software que derivan de los de usuario y a su vez se definen las pruebas de verificación que permiten comprobar que el producto cumple estos requisitos. Tras la definición de estos requisitos, se procede a realizar un diseño global de todo el sistema junto a su arquitectura por

medio de diagramas. Además, junto al diseño, se definen las pruebas de integración del sistema. A continuación, se procede a desarrollar el producto especificado y, tras la implementación, se realizan las pruebas definidas anteriormente.

3.1. Visión general

Como la cátedra se encuentra en la fase inicial, todos los detalles la misión real del nanosatélite no se han definido todavía, solo se conoce que una de sus funciones será la de tomar fotografías de la Tierra. Para poder realizar esta primera fase, se ha definido una misión sencilla. Esta misión consiste en que el software de a bordo realice un procesado periódico de los valores que obtiene de los diferentes sensores que componen el nanosatélite. Tras recoger estos valores, deberá actuar en función del valor obtenido y deberá enviar las telemetrías al ordenador de tierra para que monitorice dichos valores. El nanosatélite será capaz de: (1) obtener los valores de temperatura a través de los termistores, y en función del valor obtenido hará funcionar los calentadores, (2) calcular su altitud respecto a la tierra con los datos proporcionados por el *star tracker*, modificando su actitud, si fuese necesario, a través de sus propulsores. Además, el ordenador de a bordo contiene un mecanismo para el control de fallos que monitoriza los valores proporcionados por los sensores. Para la comunicación entre el ordenador de a bordo y el de tierra se hará uso de sockets. El ordenador de tierra contiene un servidor donde almacenará las telemetrías recibidas y fallos reportados y se comunicará con el ordenador de a bordo a través de comandos, como apagar o encender un elemento o el cambio de la frecuencia de envío de la información.

En cuanto a la órbita del nanosatélite, se ha decidido que será una órbita polar heliosíncrona. Esto significa que el nanosatélite pasará por los polos de la tierra con una inclinación de 90 grados. Que sea heliosíncrona implica que siempre tiene la misma iluminación solar.

En cuanto a la arquitectura global del sistema, el nanosatélite está compuesto por cuatro subsistemas: los periféricos del nanosatélite, el software del ordenador de a bordo, la placa hardware y el ordenador de tierra.

Nota

Este proyecto se centra en el hardware del nanosatélite y de la integración del software del ordenador de a bordo en el mismo, realizando las fases de verificación *Software-in-the-loop* y *Hardware-in-the-loop*.

En esta primera fase, en la cátedra no se dispone completamente de las especificaciones hardware, por lo que la mayoría de los componentes serán simulados. Sin embargo, pese a que tampoco se ha especificado la placa final sobre la que se va a montar el sistema, en este proyecto se ha decidido desarrollar la integración en una placa BeagleBoard-XM, tal y como se ha mencionado en el final de la sección 2.3. BealgeBoard-XM es una placa de desarrollo con un hardware similar al que probablemente se vaya a usar en el hardware final. Para comenzar el proyecto, se ha supuesto que el nanosatélite está compuesto por un conjunto de actuadores y sensores, que se dividen en dos ramas en función de la implementación. En la primera rama se encuentra la simulación de la posición del nanosatélite en el espacio. Para ello, se va a hacer uso de la herramienta *Celestia*, que permite obtener los valores de un nanosatélite simulado a través de sockets. En la segunda rama se incluyen los periféricos del nanosatélite, los cuales van a ser simulados a través del *framework* cFE para la fase *Software-in-the-loop* mientras que para la

fase *Hardware-in-the-loop* se usará hardware real que será conectado a la placa de desarrollo. De igual forma, el nanosatélite estará compuesto de placas solares que son usadas para proporcionar energía al mismo. El nanosatélite siempre estará apuntando a la tierra excepto si se produce algún error. En ese caso, apunta al sol para obtener una mayor cantidad de energía a través de sus paneles.

En la Figura 3.2 se puede observar un esquema del sistema donde se pueden diferenciar los diferentes módulos y componentes de la misión.

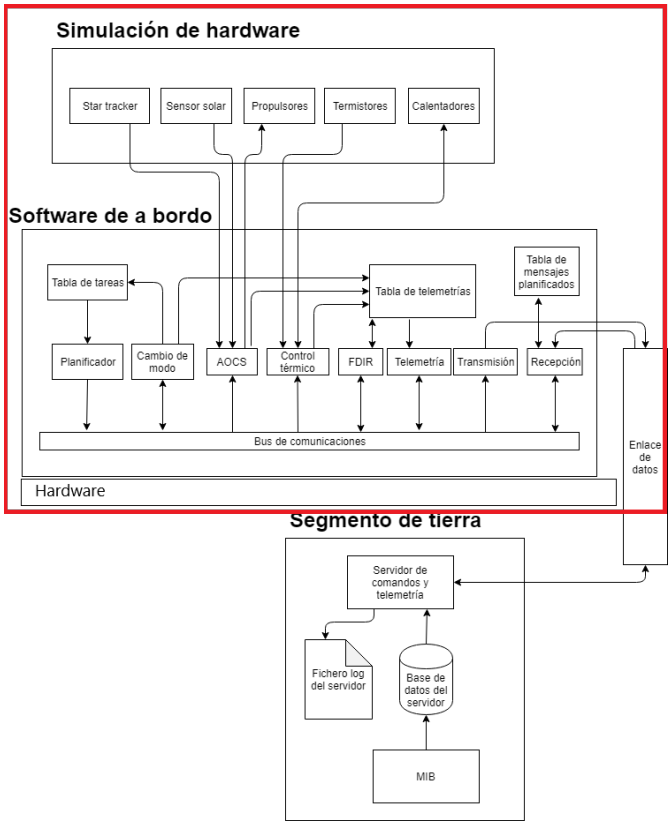


Figura 3.2: Esquema de la arquitectura del proyecto global (esta imagen se basa en la desarrollada por Pablo Andreu Sedeño, jefe de proyecto)

En la figura anterior se muestra el esquema del proyecto global. En este proyecto solo se van a tener en cuenta los módulos simulación de hardware, software de a bordo y hardware (situados dentro del recuadro), puesto que consiste en integrar dichos módulos en un sistema empotrado y verificar su funcionamiento a través de las fases *software-in-the-loop* y *hardware-in-the-loop*. A continuación se muestra una breve explicación de los diferentes elementos que componen cada módulo.

El primer módulo contiene los siguientes elementos:

Star tracker. Su objetivo es determinar el apuntamiento del nanosatélite. Consiste en una cámara integrada en la parte trasera. Esta devuelve un cuaternión [1] respecto a un eje inercial. La simulación de este módulo se realiza a través de imágenes obtenidas mediante la herramienta *Celestia*.

Sensor solar. Sensor que está formado por diversos paneles que captan el potencial eléctrico

[1] El cuaternión representa la orientación y las rotación del nanosatélite en tres dimensiones.

recibido del sol. De esta manera se puede calcular la posición del sol respecto al nanosatélite.

Propulsores. Actuadores que se accionan cuando se desea modificar la altitud del nanosatélite. Este actuador será simulado.

Termistores. Sensores que se encargar de obtener el valor de la temperatura. El sensor será simulado a través de una onda sinusoidal entre un rango de valores de temperatura máxima y mínima.

Calentadores. Actuadores que se activan en función de la temperatura. Permiten aumentar la temperatura de la zona del nanosatélite en el que se encuentren. Hay un calentador por cada termistor. Se simulará su funcionamiento, cuando estén encendidos se incrementará el valor de temperatura del sensor termistor asociado al calentador activado.

El segundo módulo contiene los siguientes elementos:

Tabla de tareas. Contiene las tareas que debe ejecutar el planificador en el orden estipulado. Se encuentra almacenada en la memoria del nanosatélite.

Tabla de telemetrías. Incluye las telemetrías generadas en el nanosatélite que son mandadas al ordenador de tierra. Se encuentra almacenada en la memoria del nanosatélite

Planificador. Se encarga de gestionar las tareas que se le ordena a través del bus de comunicaciones. Consiste en un planificador cíclico que lee de la tabla de tareas y manda un mensaje a la tarea correspondiente para que comience su ejecución.

Cambio de modo. Se encarga de cambiar entre los dos modos de ejecución del nanosatélite: el nominal, donde se apunta a la Tierra, y el seguro, donde se apunta al sol.

AOCS. Se encarga de controlar la actitud y órbita del nanosatélite. Recibe el cuartenión del *star tracker* y lo transforma a los ejes del nanosatélite, por si es necesaria una modificación de la actitud. Cuando se realiza un cambio de modo, el AOCS hace uso de los propulsores para cambiar su orientación según el modo al que se cambia.

Control térmico. Se encarga de monitorizar los valores obtenidos por los termistores. Si el valor baja del umbral establecido, pone en funcionamiento el calentador asociado. Almacena los valores de temperatura y el estado de los calentadores en la tabla de telemetría.

FDIR. Se encarga de la detección, aislamiento y recuperación de fallos. Monitoriza que los valores almacenados en la tabla de telemetría se encuentran dentro de los rangos establecidos y que no se produzcan variaciones mayores a un valor especificado. Cuando el error se produce menos de un número determinado de veces no realiza ninguna acción. Si supera dicho número, el FDIR introduce el error en el campo de error de la telemetría para que sea enviado al ordenador de tierra.

Transmisión. Se encarga de enviar periódicamente la información almacenada en la tabla de telemetrías al ordenador de tierra a través del enlace de datos.

Recepción. Se encarga de recibir los comandos enviados por el ordenador de tierra para que los envíe por el bus de comunicaciones.

El modúlo *Hardware* se encarga de la ejecución del software de a bordo junto con la simulación hardware. Por lo tanto, el módulo contiene el System-on-a-Chip en el cual se cargará el software para ser ejecutado. Este módulo abarca tanto la verificación *Software-in-the-loop* como la verificación *Hardware-in-the-loop*.

Como se ha mencionado anteriormente, este proyecto se encarga de integrar los dos módulos descritos en un sistema empotrado.

3.2. Requisitos de usuario

Existen dos tipos de requisitos de usuario.

Requisitos de capacidad. Especifican que tareas debe ser capaz de realizar la solución propuesta.

Requisitos de restricción. Especifican las restricciones aplicadas sobre los requisitos de capacidad.

Los requisitos se especifican haciendo uso de la plantilla de la Figura 3.3, donde:

Identificador. Define el formato en el cual el requisito será identificado. Será de la forma XX-UR-YY donde XX puede ser CA (de capacidad) o RE (de restricción), UR identifica que es un requisito de usuario, YY es un número de secuencia que tiene comienzo en 01.

Descripción. Una breve especificación del requisito haciendo uso de un lenguaje no ambiguo.

Necesidad. La prioridad del requisito para el cliente. Esta tomará el valor *esencial, conveniente o opcional*.

Prioridad. La prioridad para el desarrollador. Esta tomará el valor *alta, media o baja*.

Estabilidad. Indica si varía durante el proceso de desarrollo. Esta tomará el valor *no cambia, cambiante o muy inestable*.

Verificabilidad del requisito. Toma el valor *alta, media o baja*.

XX-UR-YY
Descripción:
Necesidad:
Prioridad:
Estabilidad:
Verificabilidad:

Figura 3.3: Plantilla de los requisitos de usuario

Requisitos de capacidad

CA-UR-01

Descripción:	El nanosatélite contiene un sensor que controla la altitud del mismo respecto a la Tierra.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

CA-UR-02

Descripción:	El nanosatélite contiene cuatro sensores que miden la temperatura interna de diferentes zonas del mismo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	Cambiante
Verificabilidad:	Alta

CA-UR-03

Descripción:	El nanosatélite contiene cuatro actuadores para aumentar la temperatura interna.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	Cambiante
Verificabilidad:	Alta

CA-UR-04

Descripción:	El nanosatélite contiene un sistema de localización con al sol.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Media

CA-UR-05

Descripción:	El nanosatélite contiene un un sistema de abastecimiento eléctrico.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

CA-UR-06

Descripción:	El hardware debe acomodar posibles extensiones en el número de sensores/actuadores.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Media

CA-UR-07

Descripción:	La infraestructura software/hardware será tal que soporte la ejecución del framework NASA cFE.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-03

Descripción:	El sistema operativo en tiempo real que es ejecutado sobre el hardware debe soportar la ejecución del framework NASA cFE.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

Requisitos de restricción

RE-UR-01

Descripción:	La arquitectura hardware deberá ser ARM.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Media

RE-UR-02

Descripción:	El hardware final debe soportar las condiciones del espacio exterior. Sin embargo, a efectos de simulación en Tierra, se podrá usar una placa de arquitectura/características similares que no haya pasado un proceso de certificación.
Necesidad:	Opcional
Prioridad:	Baja
Estabilidad:	Cambiante
Verificabilidad:	Alta

RE-UR-04

Descripción:	El sistema en tiempo real debe asegurar que las tareas se cumplen en el tiempo determinado.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-05

Descripción:	El gestor de arranque del sistema debe ser lo menos pesado posible.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Media

RE-UR-06

Descripción:	El gestor de arranque del sistema debe poder inicializar correctamente la placa.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-07

Descripción:	Todos los sensores deben estar conectados a la placa hardware.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-08

Descripción:	Todos los actuadores deben estar conectados a la placa hardware.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-09

Descripción:	El software del nanosatélite debe estar integrado en una placa hardware.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-10

Descripción:	El software del nanosatélite es ejecutado sobre un sistema en tiempo real.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-11

Descripción:	Todos los módulos del nanosatélite deben funcionar en la misma placa.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta

RE-UR-12

Descripción:	Para la elección del hardware final se ha de tener en cuenta la posibilidad de emular el modelo de máquina.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Media

3.3. Casos de uso

En esta sección se describen los casos de uso del sistema descrito. Un caso de uso sirve para describir el orden de los pasos que un usuario debe ejecutar para hacer uso de una funcionalidad. Los casos de uso se pueden representar a través de diagramas UML. El diagrama UML correspondiente al los casos de uso de este proyecto se puede ver en la Figura 3.4.

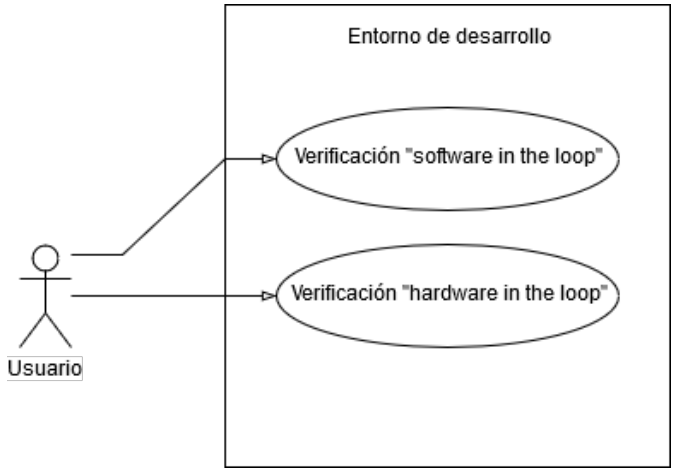


Figura 3.4: Diagrama UML de los casos de uso

Los casos de uso se especifican siguiendo el modelo descrito en la Figura 3.5. Donde:

Identificador. Define el formato en el cual el caso de uso será identificado. Será de la forma UC-XX donde UC representa que es un caso de uso, XX es un número de secuencia que tiene comienzo en 01.

Nombre. Descripción breve del caso de uso.

Actores. Rol que ejecuta el caso de uso

Objetivo del caso de uso.

Descripción. Especificación de la ejecución del caso de uso haciendo uso del lenguaje natural.

Pre-condición. Condiciones previas que se verifican antes de ejecutar el caso de uso.

Post-condición. Condiciones que se verifican después de la ejecución del caso de uso.

UC-XX

Nombre:

Actores:

Objetivo:

Descripción:

Pre-condición:

Post-condición:

Figura 3.5: Plantilla de los casos de uso

UC-01

Nombre:	Verificación Software in the loop
Actores:	Usuario
Objetivo:	Verificar que una imagen de disco puede ser ejecutada en un emulador de hardware.
Descripción:	Se ejecuta una imagen de disco en el emulador de hardware. Cuando arranca el emulador la imagen de disco es ejecutada. La imagen contiene el software de a bordo del nanosatélite junto con la simulación de los sensores y actuadores del mismo.
Pre-condición:	El emulador de hardware tiene soporte para BeagleBoard-XM. Tener generada una imagen de disco con soporte para la máquina BeagleBoard-XM.
Post-condición:	Se ha ejecutado correctamente la imagen de disco.

UC-02

Nombre:	Verificación hardware in the loop
Actores:	Usuario
Objetivo:	Verificar que una imagen de disco puede ser ejecutada en hardware.
Descripción:	Se inserta la imagen de disco en el hardware y se enciende dicha placa para que la imagen sea ejecutada. La imagen de disco contiene el software de a bordo. Los periféricos del nanosatélite son conectados al hardware mediante los puertos GPIO habilitados.
Pre-condición:	Tener generada una imagen de disco con soporte para la máquina BeagelBoard-XM.
Post-condición:	Se ha ejecutado correctamente la imagen de disco.

3.4. Requisitos de software

En esta sección se incluye la especificación de requisitos de software. Estos se han obtenido a partir de los requisitos de usuario de la Sección 3.2. Describen la interpretación del analista sobre los requisitos de usuario. Los requisitos de software pueden ser de dos tipos:

Requisitos funcionales. Especifican las características funcionales del software.

Requisitos no funcionales. Especificaciones adicionales que no añaden una nueva funcionalidad (de tiempo, de rendimiento…).

El formato de los requisitos de software se define a tal y como se muestra en la Figura 3.6, donde:

Identificador. Define el formato en el cual el requisito será identificado. Tiene el formato XX-YY-ZZ donde XX puede ser RF (si es requisito funcional) o NF (si es un requisito no funcional). YY describirá los tipos de requisitos, SR (para requisitos funcionales y para no funcionales). ZZ es un número de secuencia que tiene comienzo en 01.

Necesidad. La prioridad del requisito para el cliente. Esta tomará el valor *esencial, conveniente o opcional*.

Prioridad. La prioridad para el desarrollador. Esta tomará el valor *alta, media o baja*.

Estabilidad. Indica si varía durante el proceso de desarrollo. Esta tomará el valor *no cambia, cambiante o muy inestable*.

Verificabilidad del requisito. Toma el valor *alta, media o baja*.

Origen. Hace referencia a que requisitos de usuario originaron este requisito.

XX-YY-ZZ
Descripción:
Necesidad:
Prioridad:
Estabilidad:
Verificabilidad:
Origen:

Figura 3.6: Plantilla de los requisitos de software

Requisitos funcionales

RF-SR-01

Descripción:	El sensor que controla el apuntamiento del nanosatélite respecto a la tierra consiste en una cámara integrada en la parte trasera.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-01

RF-SR-02

Descripción:	El sensor de apuntamiento devuelve el cuaternión de la zona del espacio a la que apunta.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Media
Origen:	CA-UR-01

RF-SR-03

Descripción:	Los sensores de medición de temperatura interna son sensores térmicos integrados en diferentes partes del nanosatélite.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-02

RF-SR-04

Descripción:	El sensor de temperatura devuelve el valor de la temperatura de la zona del nanosatélite que mide. El valor devuelto se encuentra en grados Kelvin.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-03

RF-SR-05

Descripción:	La temperatura interna del nanosatélite deberá ser incrementada por unos calentadores.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-03

RF-SR-06

Descripción:	El calentador se activa cuando la temperatura interna se encuentra por debajo del límite inferior.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	CA-UR-03

RF-SR-07

Descripción:	El calentador se desactiva cuando la temperatura interna supera el límite superior, siempre y cuando esté activado.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	CA-UR-03

RF-SR-08

Descripción:	Cada sensor térmico tiene asociado un calentador.
Necesidad:	Conveniente
Prioridad:	Baja
Estabilidad:	Cambiante
Verificabilidad:	Baja
Origen:	CA-UR-02, CA-UR-03

RF-SR-09

Descripción:	El sistema de localización del sol respecto al nanosatélite está compuesto por una rejilla de paneles solares.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-04

RF-SR-10

Descripción:	Los paneles solares devuelven el potencial eléctrico de la energía solar que incide sobre él.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-04

RF-SR-11

Descripción:	El sistema de abastecimiento eléctrico se basa en la captación de la energía solar por medio de placas solares.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-05

RF-SR-12

Descripción:	Las placas solares consisten en celdas captan la energía solar y la transforman en potencia elétrica.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-05

RF-SR-13

Descripción:	El hardware debe tener suficientes puertos donde conectar todos los sensores/actuadores. Además, debe tener puertos extra para poder añadir otros sensores/actuadores si así se desea.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Media
Origen:	CA-UR-06

RF-SR-14

Descripción:	En el hardware se debe poder integrar el framework NASA cFE.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	CA-UR-07

RF-SR-15

Descripción:	El framework NASA CFe es la plataforma usada para simular los sensores térmicos.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Media
Origen:	CA-UR-02, CA-UR-03, CA-UR-07

RF-SR-16

Descripción:	El framework NASA CFe es la plataforma usada para simular los calentadores.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Media
Origen:	CA-UR-02, CA-UR-03, CA-UR-07

RF-SR-17

Descripción:	El framework NASA CFe es la plataforma usada para simular los sensores solares.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Media
Origen:	CA-UR-02, CA-UR-03, CA-UR-07

Requisitos no funcionales

NF-SR-01

Descripción:	Sobre el hardware se ejecuta el sistema en tiempo real que contiene el software de a bordo.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-03, RE-UR-04, RE-UR-10

NF-SR-02	
Descripción:	El sistema operativo en tiempo real que es ejecutado sobre el hardware es RTEMS.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	RE-UR-03, RE-UR-04, RE-UR-10

NF-SR-03	
Descripción:	RTEMS debe de estar compilado para la arquitectura ARM.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	RE-UR-01, RE-UR-03, RE-UR-04, RE-UR-10

NF-SR-04	
Descripción:	La instalación de RTEMS debe tener soporte para la máquina BeagleBoard-XM.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	RE-UR-01, RE-UR-03, RE-UR-04, RE-UR-10

NF-SR-05	
Descripción:	Para ejecutar las aplicaciones del sistema empotrado se necesita un gestor de arranque.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-05, RE-UR-06

NF-SR-06	
Descripción:	El gestor de arranque a utilizar es das U-Boot.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-05, RE-UR-06

NF-SR-07	
Descripción:	El gestor de arranque del sistema debe estar compilado para una arquitectura ARM.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	RE-UR-01, RE-UR-05, RE-UR-06

NF-SR-08

Descripción:	El gestor de arranque del sistema debe tener soporte para la máquina BeagleBoard-XM.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	RE-UR-01, RE-UR-05, RE-UR-06

NF-SR-09

Descripción:	El software de emulación es QEMU.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-12

NF-SR-10

Descripción:	QEMU debe tener soporte para BeagleBoard-XM.
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	Cambiante
Verificabilidad:	Alta
Origen:	RE-UR-12

NF-SR-11

Descripción:	Todos los sensores deben estar conectados a la placa hardware por medio de los puertos habilitados en el hardware.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-07

NF-SR-12

Descripción:	Todos los actuadores deben estar conectados a la placa hardware por medio de los puertos habilitados en el hardware.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-08

NF-SR-13

Descripción:	Sobre la placa hardware debe funcionar correctamente las aplicaciones software junto con los sensores y actuadores.
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-07, RE-UR-08, RE-UR-09, RE-UR-10, RE-UR-11

NF-SR-14

Descripción:	Para la ejecución en el espacio, el hardware debe ser capaz de soportar las condiciones extremas a las que se expone en el espacio exterior.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-02

NF-SR-15

Descripción:	El hardware debe poder soportar la radiación electromagnética que proviene en el espacio.
Necesidad:	Esencial
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-02

NF-SR-16

Descripción:	Para la simulación en Tierra se puede usar hardware que no soporte las condiciones del espacio exterior.
Necesidad:	Opcional
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Baja
Origen:	RE-UR-02

NF-SR-17

Descripción:	El software del ordenador de a bordo debe ser verificado a través del procedimiento Software in the loop
Necesidad:	Conveniente
Prioridad:	Media
Estabilidad:	No cambia
Verificabilidad:	Media
Origen:	RE-UR-12

NF-SR-18

Descripción:	El software del ordenador de a bordo debe ser verificado a través del procedimiento Hardware in the loop
Necesidad:	Esencial
Prioridad:	Alta
Estabilidad:	No cambia
Verificabilidad:	Alta
Origen:	RE-UR-11

3.5. Diseño de la solución

Nota

El proyecto consta de dos fases de verificación:

Verificación *Software-in-the-loop*. En esta fase se verifica el funcionamiento del sistema empotrado sobre una simulación del hardware.

Verificación *Hardware-in-the-loop*. En esta fase se verifica el funcionamiento del sistema empotrado sobre el hardware real.

Como se ha mencionado anteriormente, la Figura 3.2 describe la arquitectura general del nanosatélite. La arquitectura de este proyecto se puede ver en las Figuras 3.7 y 3.8, donde la Figura 3.7 muestra los diferentes componentes de la arquitectura general que interactúan con la emulación del software y la Figura 3.8 muestra los diferentes componentes de la arquitectura general que interactúan con el hardware.

Arquitectura para *software-in-the-loop*

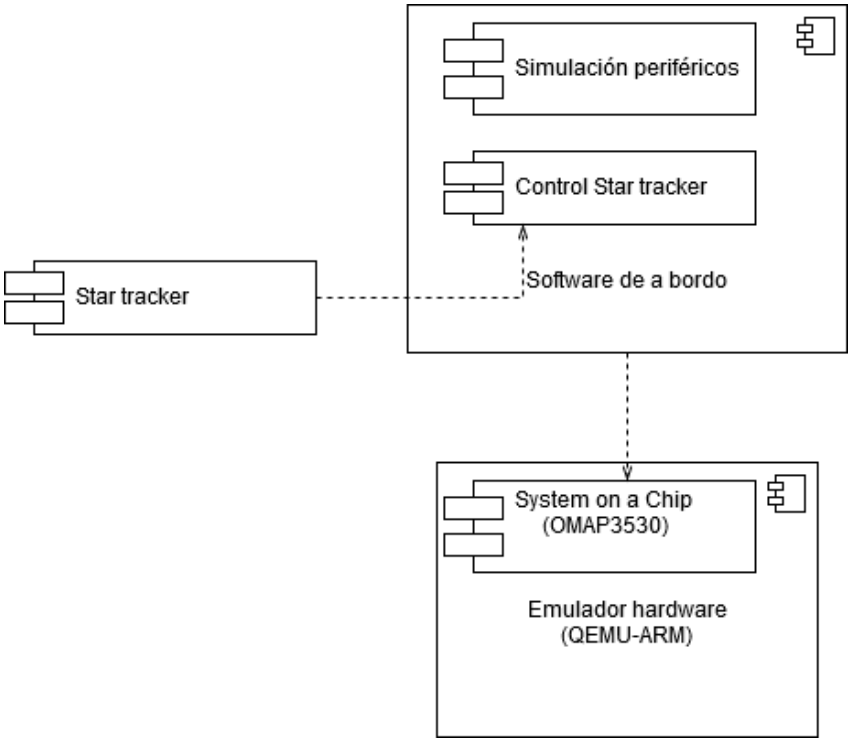


Figura 3.7: Arquitectura emulación hardware

Arquitectura para *hardware-in-the-loop*

A continuación se muestran las tablas de componentes correspondientes a los dos diagramas anteriores. Para su definición se va a usar la plantilla de la Figura 3.9, donde:

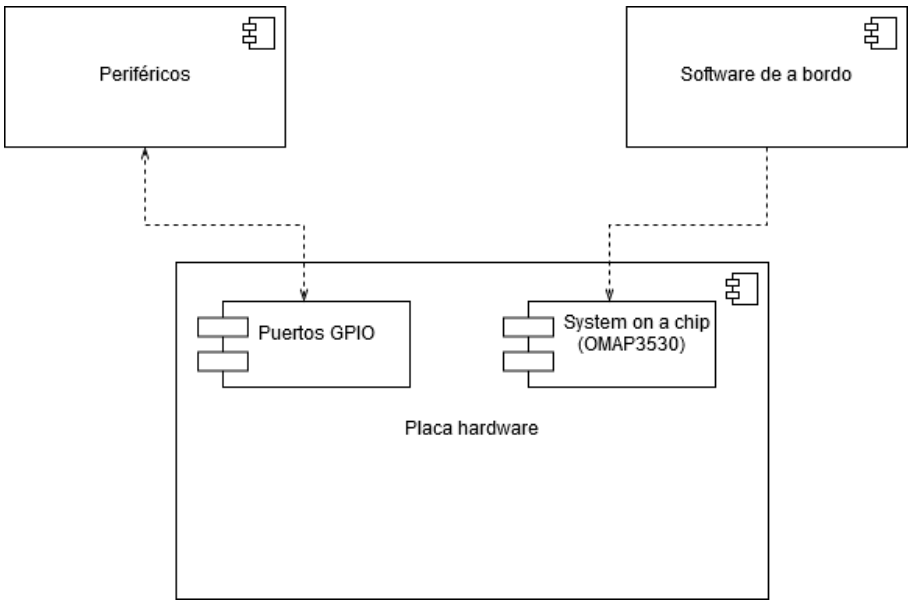


Figura 3.8: Arquitectura hardware

Identificador. Nombre identificativo del componente. En este caso, será en nombre que se encuentra dentro del componente de las Figuras 3.7 y 3.8.

Rol del componente en el proyecto.

Dependencias. Que componentes dependen de este.

Descripción. Descripción breve del comportamiento del componente.

Datos. Entradas y salidas del componente.

Recursos utilizados por el componente.

Origen. Requisitos software que producen el componente.

Identificador	
Rol:	
Dependencias:	
Descripción:	
Datos:	n/a
Recursos:	n/a
Origen:	

Figura 3.9: Plantilla de los componentes

Star tracker

Rol: Devolver la posición en el espacio.

Dependencias:

Descripción: El periférico calcula la posición en el espacio tomando como referencia un mapa de las estrellas. Capta una imagen de la zona en la que se encuentra y la compara con el mapa estelar. Se trata de un componente externo puesto que se simula con la herramienta externa *Celestia*. La comunicación entre este componente y el software de a bordo se realiza a través de sockets.

Datos:

- [in]: Imagen del espacio.
- [out]: Cuaternión con la posición.

Recursos: n/a

Origen: RF-SR-01, RF-SR-02

Control star tracker

Rol: Transformar el cuaternión al eje de coordenadas.

Dependencias: Software de a bordo

Descripción: El componente, al transformar el cuaternión envía la información a la tabla de telemetrías.

Datos:

- [in]: Cuaternión con la posición.
- [out]: Posición en el eje de coordenadas.

Recursos: n/a

Origen: RF-SR-01, RF-SR-02

Simulación de periféricos

Rol: Componente que agrupa las simulaciones de los periféricos.

Dependencias: Software de a bordo

Descripción: Los periféricos simulados son el sensor térmico, sensor solar, propulsor y calentador.

Datos:

- [in]: Activar/desactivar calentador y activar/desactivar propulsor.
- [out]: Valores de temperatura en kelvin; potencial eléctrico; estado del calentador.

Recursos: n/a

Origen: RF-SR-03, RF-SR-04, RF-SR-05, RF-SR-06, RF-SR-07, RF-SR-08, RF-SR-09, RF-SR-10, RF-SR-11, RF-SR-12, RF-SR-15, RF-SR-16, RF-SR-17

Periféricos

Rol: Componente que engloba los periféricos del nanosatélite.

Dependencias:

Descripción: Los periféricos que conforman el nanosatélite son sensor térmico, sensor solar, propulsor, calentador y star tracker. Se conectan al hardware a través de los puertos GPIO habilitados.

Datos:

- [in]: Temperatura; potencial eléctrico; activar/desactivar calentador; activar/desactivar propulsor. [out]: Valores de temperatura en kelvin; potencial eléctrico; estado del calentador.

Recursos: n/a

Origen: RF-SR-03, RF-SR-04, RF-SR-05, RF-SR-06, RF-SR-07, RF-SR-08, RF-SR-09, RF-SR-10, RF-SR-11, RF-SR-12, RF-SR-13

Software de a bordo

Rol: Se encarga de monitorizar los datos enviados por los periféricos y actuar en consecuencia.

Dependencias:

Descripción: Cuando recibe un valor lo analiza, lo empaqueta en un paquete y lo almacena la tabla de telemetrías. Para cada tipo de valor existe un paquete diferente. Tanto la simulación de los sensores como el software de a bordo ha sido desarrollado por compañeros de la cátedra.

Datos:

- [in]: Valores de temperatura en kelvin; potencial eléctrico; estado del calentador.
- [out]: Paquete de térmico; paquete solar; activar/desactivar calentador; activar/desactivar propulsor.

Recursos: n/a

Origen: RF-SR-04, RF-SR-06, RF-SR-10

Emulador hardware

Rol: Se encarga de ejecutar el software de a bordo emulando un hardware.

Dependencias:

Descripción: Simula las especificaciones del hardware para la fase de verificación *Software-in-the-loop*, donde el software de a bordo es ejecutado. El emulador hardware utilizado es QEMU.

Datos:

- [in]: Arrancar simulación.
- [out]: Ejecución del ordenador de a bordo.

Recursos: n/a

Origen: RF-SR-14

Placa hardware

Rol: Se encarga de ejecutar el software de a bordo.

Dependencias:

Descripción: El hardware es el componente en el cual será ejecutado el software de a bordo para la fase de verificación *Hardware-in-the-loop*. Incluye puertos GPIO para poder conectar los periféricos.

Datos:

- [in]: Encender placa.
- [out]: Ejecución del ordenador de a bordo.

Recursos: n/a

Origen: RF-SR-13

System on a Chip(OMAP3530)

Rol: Componente en el que es integrado el software de a bordo.

Dependencias: Emulador hardware, Placa hardware

Descripción: El System on a Chip usado en este proyecto pertenece a la familia OMAP3530 de *Texas Instruments*.

Datos:

- [in]: Software de a bordo.
- [out]: Ejecución del software de a bordo.

Recursos: n/a

Origen: RF-SR-14, RF-SR-15 RF-SR-16, RF-SR-17

Puertos GPIO

Rol:	Conectar los periféricos con el hardware.
Dependencias:	Placa hardware
Descripción:	Se dispone de suficientes puertos GPIO en la BeagleBoard-XM como para conectar los periféricos. Cada periférico usa un número variable de pines GPIO.
Datos:	▪ [in]: Periféricos. ▪ [out]: Valores de temperatura en Kelvin; potencial eléctrico; estado del calentador.
Recursos:	n/a
Origen:	RF-SR-13

3.6. Entorno de desarrollo

Respecto al entorno de desarrollo del proyecto se han tomado las siguientes decisiones:

- La arquitectura de la plataforma hardware es ARM. Su elección se debe a que tiene una amplia gama de microprocesadores que soportan un RTOS y el consumo de los mismos es relativamente bajo. Además, es una arquitectura comúnmente usada en las misiones espaciales puesto que ofrece un alto rendimiento en sistemas empotrados y para sistemas de tiempo real.

- El *framework* usado para el desarrollo de la simulación de los módulos del nanosatélite es cFE. Esto se debe a que es un *framework* de código abierto que ha sido utilizado en múltiples misiones. Además, permite que los diferentes módulos del nanosatélite puedan compilarse, cargarse e iniciarse sin tener que reconstruir todo el sistema. cFE permite que cada módulo pueda ser probado independientemente y, tras comprobar su correcto funcionamiento, integrar todos los módulos en un sistema empotrado sin necesidad de tener que modificar el software.

- El RTOS sobre el que se ejecutan los módulos del nanosatélite es RTEMS. RTEMS ofrece completo soporte para el *framework* cFE. Además, se trata de un sistema *open source* con soporte para la mayoría de arquitecturas hardware, entre ellas ARM. RTEMS está diseñado para sistemas empotrados que requieren una respuesta rápida, una cierta seguridad y estabilidad.

- El gestor de arranque de la imagen de disco es U-Boot. Se ha decidido usar U-Boot pues es el sistema de arranque usado en la mayoría de sistemas empotrados debido a que se trata de un sistema poco pesado y permite completar el proceso de iniciación del hardware. Además, U-boot tiene soporte para la arquitectura ARM.

3.7. Matrices de trazabilidad

En las siguientes páginas se muestran las matrices de trazabilidad. En la Figura 3.10 se muestra la matriz de trazabilidad entre los requisitos de capacidad y los requisitos funcionales. En la Figura 3.11 se muestra la matriz de trazabilidad entre los requisitos de restricción y los requisitos no funcionales. En la Figura 3.12 se muestra la matriz de trazabilidad entre los componentes y los requisitos funcionales.

	CA-UR-01	CA-UR-02	CA-UR-03	CA-UR-04	CA-UR-05	CA-UR-06	CA-UR-07
RF-SR-01	•						
RF-SR-02	•						
RF-SR-03		•					
RF-SR-04			•				
RF-SR-05			•				
RF-SR-06			•				
RF-SR-07			•				
RF-SR-08		•	•				
RF-SR-09				•			
RF-SR-10				•			
RF-SR-11					•		
RF-SR-12					•		
RF-SR-13						•	
RF-SR-14							•
RF-SR-15		•	•				•
RF-SR-16		•	•				•
RF-SR-17		•	•				•

Figura 3.10: Matriz de trazabilidad: requisitos de capacidad - requisitos funcionales

	RE-UR-01	RE-UR-02	RE-UR-03	RE-UR-04	RE-UR-05	RE-UR-06	RE-UR-07	RE-UR-08	RE-UR-09	RE-UR-10	RE-UR-11	RE-UR-12
NF-SR-01			•	•						•		
NF-SR-02			•	•						•		
NF-SR-03	•		•	•						•		
NF-SR-04	•		•	•						•		
NF-SR-05					•	•						
NF-SR-06					•	•						
NF-SR-07	•				•	•						
NF-SR-08	•				•	•						
NF-SR-09												•
NF-SR-10												•
NF-SR-11							•					
NF-SR-12								•				
NF-SR-13							•	•	•	•	•	
NF-SR-14		•										
NF-SR-15		•										
NF-SR-16		•										
NF-SR-17												•
NF-SR-18											•	

Figura 3.11: Matriz de trazabilidad: requisitos de restricción - requisitos no funcionales

	RF-SR-01	RF-SR-02	RF-SR-03	RF-SR-04	RF-SR-05	RF-SR-06	RF-SR-07	RF-SR-08	RF-SR-09	RF-SR-10	RF-SR-11	RF-SR-12	RF-SR-13	RF-SR-14	RF-SR-15	RF-SR-16	RF-SR-17
Control star tracker	•	•															
Emulador hardware													•				
Periféricos			•	•	•	•	•	•	•	•	•	•					
Placa hardware												•					
Puertos GPIO												•					
Simulación de periféricos			•	•	•	•	•	•	•	•	•			•	•	•	
Software de a bordo				•		•				•							
Star tracker	•	•															

Figura 3.12: Matriz de trazabilidad: componentes - requisitos software

Capítulo 4

Implementación y pruebas

En este capítulo se describen detalles de algunos de los procesos implementados durante el desarrollo del proyecto. El resultado de la implementación consiste en integrar en un sistema empotrado una imagen de disco que al arrancar ejecute un firmware basado en RTEMS. El desarrollo de la implementación de este proyecto se ha realizado para una máquina tipo BeagleBoard-XM. Como se ha mencionado en el Capítulo 3, el proyecto consta de dos fases de verificación: la verificación *Software-in-the-loop*, donde se verifica el funcionamiento sistema en un entorno simulado del hardware, y la verificación *Hardware-in-the-loop*, donde se verifica el sistema final en hardware físico.

Además, en este capítulo se incluyen las pruebas realizadas para comprobar el correcto funcionamiento del software desarrollado así como su evaluación de rendimiento.

4.1. Estudio de viabilidad del sistema

Antes de generar una imagen de disco que contiene un sistema empotrado que incluye el sistema operativo RTEMS, se ha decidido desarrollar una imagen de disco que contiene un sistema empotrado con el sistema operativo Linux. Al arrancar el sistema empotrado, se ejecuta BusyBox, un programa que combina muchas utilidades estándar de Unix en su ejecutable. Se ha decidido generar esta imagen para estudiar la viabilidad del sistema, ya que al ser una prueba sencilla que no lleva mucho tiempo de desarrollo podremos comprobar si es posible generar el sistema empotrado con RTEMS. La imagen de disco del sistema empotrado será generada para su ejecución en el hipervisor VMWare.

Antes de realizar el proceso de generación de la imagen de disco, se tiene que preparar:

El directorio etc/. El directorio `etc/` contiene la configuración de todos los programas que son ejecutados en la máquina. En este caso, contiene configuración para el arranque e información sobre las redes de conexión. El directorio es comprimido en un archivo *.tgz*.

Directorio kernel/. Este directorio contiene la configuración del kernel que se va a instalar. Contiene el arranque a través del gestor *GRUB* y las librerías necesarias. El directorio es comprimido en un archivo *.tgz*.

El desarrollo de la imagen se ha estructurado en cuatro scripts:

- `create_imagen.sh`: Este script contiene el proceso de creación de una imagen del disco vacía. Para ello, se crea un fichero vacío con tamaño 1000MB. A continuación, se genera la tabla de particiones y se asigna la imagen a un dispositivo virtual por medio del mapeo de las particiones. Después, se procede a generar un sistema de ficheros en la partición anterior, donde se monta un directorio temporal en el que se crea el archivo de configuración de GRUB, el sistema de arranque usado por Linux. Tras haber generado el archivo de configuración de GRUB se configura el mismo para el arranque de la imagen. Se desmonta el directorio temporal, se libera el dispositivo virtual y se borra el directorio temporal.

- `create_busybox.sh`: Este script se encarga de integrar BusyBox en el disco. Para poder instalar BusyBox en el disco generado anteriormente, se realiza la asignación de la imagen a un dispositivo virtual de la misma forma que en la fase anterior. Se un directorio temporal que es montado en el sistema de ficheros que se ha creado en el script anterior. Dentro del directorio, se generan los directorios */dev, /proc, /sys, /var/run y /bin*. Es en este último donde se copia el software de BusyBox. Tras esto, se crean los enlaces de todos los programas que contienen BusyBox. En el directorio se descomprime el archivo de configuración del etc. Se desmonta el directorio temporal, liberamos el dispositivo virtual y borramos el directorio temporal. Se desmonta el directorio temporal, se libera el dispositivo virtual y se borra el directorio temporal.

- `create_kernel.sh`: Este script se encarga de integrar el kernel de Linux en el disco. El kernel es el responsable de facilitar a los programas el acceso seguro al hardware. Es el responsable de gestionar recursos a través de servicios de llamadas al sistema. Para introducir el kernel en el disco, se realiza la asignación de la imagen a un dispositivo virtual. Se crea un directorio temporal que es montado en el sistema de ficheros generado anteriormente. En el directorio temporal se descomprime el fichero `kernel.tgz`, que contiene el kernel Linux. Se desmonta el directorio temporal, se libera el dispositivo virtual y se borra el directorio temporal.

- `convert_imagen.sh`: Este script se encarga de convertir la imagen de disco a formato *vmdk* para que pueda ser importada a una máquina virtual de *VMware*.

Algunos de los comandos ejecutados por los anteriores scripts se pueden observar en el Apéndice C.2. Además, se ha generado un script que llama a los scripts anteriores, en el mismo orden que se han descrito. Todos los scripts son ejecutados con permisos de superusuario ya que que son necesarios para crear una imagen de disco de un sistema empotrado.

Ejecución del sistema empotrado

Para probar la imagen de disco se puede usar el emulador QEMU. Es resultado de la ejecución se puede ver reflejado en la Figura 4.1. También podemos probar la imagen a través de una máquina virtual de *VMWare*. Para ello, al crear una máquina virtual, se modifica el disco duro de la misma sustituyéndolo por la imagen que hemos creado anteriormente. Cuando se ejecuta la máquina virtual, arranca el sistema empotrado. En este caso, al arrancar el sistema se ejecuta el programa BusyBox, que muestra un interfaz similar a la de la terminal de Linux, tal y como se puede ver en la Figura 4.1.

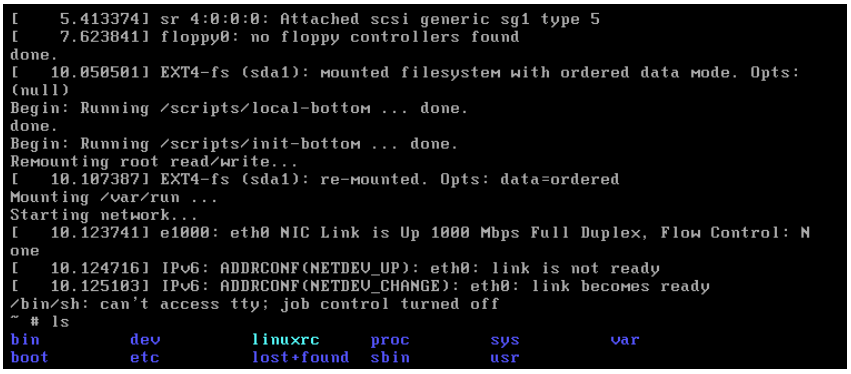


Figura 4.1: Imagen de ejecución de linux empotrado.

4.2. Configuración del entorno

En esta sección se incluyen algunos detalles sobre el proceso de configuración del entorno de desarrollo para la integración de un sistema empotrado.

RTEMS

Como se ha mencionado en el Capítulo 2, RTEMS es un sistema operativo de tiempo real libre que está diseñado para sistemas empotrados que necesitan respuesta rápida y estabilidad. La instalación de RTEMS se realiza sobre la máquina virtual. La estructura de directorios principales en RTEMS se puede ver en la Figura 4.2.

Figura 4.2: Estructura de directorios principales en RTEMS.

Donde,

$HOME/development/rtems. Es el directorio raíz de toda la estructura de desarrollo.

$HOME/development/rtems/rsb. Es el directorio donde se almacena de la herramienta *Source Builder*.

$HOME/development/rtems/5. Es el directorio donde se encuentran los ejecutables de las herramientas necesarias para compilar RTEMS en la arquitectura deseada. Esta compilación se almacena dentro de este directorio. En este caso, se realizará la compilación para ARM y se almacena dentro de la carpeta rtems-arm.

$HOME/development/rtems/kernel. Es el repositorio donde se almacena los archivos de instalación de RTEMS. Durante la instalación, se compilan estos archivos para el hardware deseado y se almacena la instalación dentro de este directorio en una carpeta específica para el hardware. En este caso, se realizará la compilación para BeagleBoard-XM y se almacena dentro de la carpeta `beagleboardxm`.

El proceso completo de instalación del sistema operativo se incluye en el Apéndice C, en la sección C.1. En los siguientes párrafos se describen algunos detalles del mismo [45].

En primer lugar, para instalar RTEMS es necesario instalar el *Source Builder*, una herramienta que permite generar los diferentes módulos de RTEMS a partir de su código fuente. El *Source Builder* permite automatizar la instalación y proporciona el conjunto de herramientas necesario para instalar el sistema operativo. La instalación almacenará en el directorio raíz, mencionado anteriormente, toda la estructura del entorno de desarrollo de RTEMS. Una vez descargada la herramienta *Source Builder*, se debe instalar el conjunto de herramientas de compilación, denominado *Build Set*. El *Build Set* instalado ha sido el de la arquitectura ARM. En el listado 4.1 se muestra el comando para instalar el *Build Set* para ARM.

Listado 4.1: Comando instalar el Build Set para ARM

```
1 $ ../source-builder/sb-set-builder --prefix=$HOME/development/rtems/5 5/rtems-arm
```

A continuación, se compila el Kernel de RTEMS después de clonar su repositorio y ejecutar el archivo de configuración mediante *bootstrap*. El kernel debe estar compilado para un hardware específico. Esto se indica a través del BSP (*Board Support Package*), que es la capa de software que especifica el hardware de los drivers que permiten la comunicación de RTEMS con el hardware específico utilizado. En el caso de este proyecto, el BSP usado es `beagleboardxm`. Además se activa la interfaz POSIX y la generación de test de prueba, entre otros. Esto se realiza a través de *flags* en el comando de instalación del kernel. Los test generados se encuentran almacenados en el directorio *$HOME/development/rtems/kernel/beagleboardxm/arm-rtems5/c/beagleboardxm/testsuites/samples*. En el listado 4.2 se muestra el comando a ejecutar para instalar RTEMS para BeagleBoard-XM.

Listado 4.2: Comando instalar el RTEMS para BeagleBoard-XM

```
1 $ $HOME/development/rtems/kernel/rtems/configure --prefix=$HOME/development/rtems/5
      --target=arm-rtems5 --enable-rtemsbsp=beagleboardxm --enable-posix -enable-
     tests=samples --enable-rtems-debug
```

Tras realizar la instalación de RTEMS, se modifica el fichero .bashrc, insertando la línea del Listado 4.3 en el fichero para mantener persistente los ficheros binarios de RTEMS.

Listado 4.3: Línea a incluir en .bashrc para mantener persistente los ficheros binarios de RTEMS

```
1 export PATH=$HOME/development/rtems/5/bin:$PATH
```

U-Boot

En arquitecturas como x86, la BIOS-ROM del sistema inicia el arranque de la máquina y, a continuación, el gestor de arranque finaliza el proceso de iniciación del sistema. Sin embargo,

en arquitecturas como ARM, la BIOS-ROM no existe o es demasiado ligera para inicializar completamente el hardware. En estas arquitecturas se pasa parcialmente el control de la iniciación de la máquina al gestor de arranque. En el caso de este proyecto, el gestor de arranque seleccionado ha sido U-Boot, debido a que cumple con las especificaciones del proyecto, es capaz de iniciar el sistema desde su arranque. Además, debido a su ligereza, es el sistema de arranque que usado en la mayoría de los sistemas empotrados.

Al igual que en el caso de RTEMS, para compilar U-Boot es necesario realizar una compilación cruzada. En este caso, al tratarse de una compilación para la arquitectura ARM, se debe instalar el paquete para compilación cruzada *arm-linux-gnueabi* mediante el comando del Listado 4.4. Este paso no s realizó de forma manual antes porque RTEMS instala y usa su propio *toolchain*.

Listado 4.4: Comando para instalar la compilación cruzada arm-linux-gnueabi

```
1 $ sudo apt-get install gcc-arm-linux-gnueabi
```

Tras clonar el repositorio de u-Boot en la carpeta deseada, se procede a configurar U-boot. Existen dos métodos:

Configuración por defecto: U-Boot contiene unos ficheros de configuración por defecto que al ser compilados, aplican la configuración determinada en el fichero a U-Boot. Para compilar el fichero se hace uso de la compilación cruzada de *arm-linux-gnueabi*. En este caso, como se desea el funcionamiento para placas tipo BeagleBoard, se configura U-boot a través de la configuración por defecto *omap3_beagle_defconfig*, ya que la arquitectura del procesador de las placas BeagleBoard es de tipo OMAP3.

Configuración personalizada: También se puede crear una configuración de U-boot. Se puede personalizar la configuración a través de la interfaz de configuración que incluye U-boot en su repositorio. Se compila el *target "menuconfs"* con la herramienta *make* y aparece el menú de la Figura 4.3. En el menú de configuración se pueden modificar los parámetros de configuración para que se adapten al hardware que se usa.

La configuración de U-Boot puede ser modificada en cualquier momento, ya sea por el primer método o por el segundo. Siempre que se cambie la configuración se tiene que recompilar U-Boot. Una vez se ha configurado U-Boot, se procede a compilar el software a través de la compilación cruzada de *arm-linux-gnueabi*. En el listado 4.5 se muestra el comando para configurar por defecto U-Boot para BeagleBoard y el comando para compilar U-Boot.

Listado 4.5: Comandos para compilar U-Boot

```
1 # Se configura U-Boot para BeagleBoard
2 $ ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- make omap3_beagle_defconfig
3
4 # Se compila U-Boot
5 $ ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- make
```

Todos los comandos necesarios para la compilación de U-Boot para placas BeagleBoard se encuentran en el Apéndice C.3.

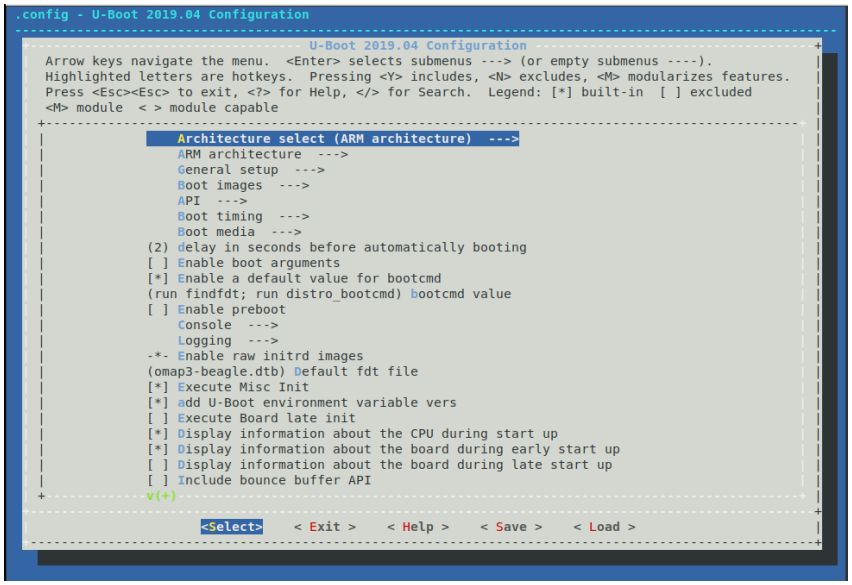


Figura 4.3: Interfaz del menú de configuración de U-Boot

Qemu-linaro

Para la fase de verificación *software-in-the-loop*, se ha utilizado el emulador *QEMU*. QEMU es un emulador de procesadores que se basa en la traducción dinámica de archivos binarios. QEMU tiene capacidades de virtualización dentro de un sistema operativo como Linux o Windows [46]. Esta máquina virtual puede ejecutarse en cualquier tipo de arquitectura tales como ARM, PowerPC y SPARC. Sin embargo, la versión *upstream* de QEMU no soporta la emulación de placas BeagleBoard, por lo que se ha tenido que instalar una extensión no oficial de QEMU que contiene el soporte para BeagleBoard. Esta extensión recibe es conocida como Qemu-linaro [47].

Para compilar Qemu-linaro es necesario compilar mediante Clang debido a que las versiones modernas de GCC (*GNU Compiler Collection*) no permiten compilar Qemu-linaro. Clang es un compilador para los lenguajes de programación C, C++, Objective-C y Objective-C++, que está diseñado como alternativa a GCC. Para instalar Clang se ejecuta el comando del listado 4.6.

Listado 4.6: Comando para instalar Clang

```
1 $ CC=clang CXX=clang++ ./configure --prefix=$HOME/opt/qemu-linaro --disable-
    smartcard-nss
```

Se clona el repositorio de Qemu-linaro en la carpeta que se desea. Antes de compilarlo, se modifica los ficheros con `qga/commands-posix.c` y `hw/9pfs/virtio-9p.c` donde se añade la siguiente línea: `include <sys/sysmacros.h>`. Por último, compilamos Qemu-linaro con *cmake*. Todos los comandos a ejecutar para compilar Qemu-linaro se encuentran en el Apéndice C.4.

4.3. Generación del firmware del sistema empotrado

Como se ha comentado, el sistema empotrado a bordo del satélite, ejecutará un firmware basado en un RTOS. Para probar el arranque del sistema, se generará una imagen que pase el control a una aplicación RTEMS de ejemplo (ticker.exe). Finalmente, se describirá el proceso completo, pasando la ejecución a un kernel basado en RTEMS+cFE.

Sistema empotrado con ejecutable de RTEMS

La imagen de disco contiene el ejecutable de RTEMS junto con el sistema de arranque (U-Boot). Como se ha mencionado en el Capítulo 2.6, U-boot se puede dividir en dos fases de arranque: primero se carga el SPL, que realiza la configuración inicial del hardware, y carga la versión completa de U-Boot. Por lo tanto, para seguir este procedimiento es necesario dos archivos. El SPL es un fichero de nombre `MLO` [1] y el fichero con las versión completa de U-boot que es un fichero con nombre u-boot.bin. Ambos ficheros son generados durante la compilación de U-boot y se encuentran en su directorio raíz. El último elemento principal para que la imagen arranque es el ejecutable compilado y vinculado con el ejecutivo RTEMS.

Como se ha mencionado anteriormente, la imagen va a ser probada en hardware OMAP3, ya sea simulado o real. Los *System-on-a-Chip* OMAP3 de *Texas Intruments* tienen dos procedimientos de arranque:

1. Se genera una tabla de particiones MBR (*Master Boot Record*) con una partición activa en FAT12/16/32. En su directorio raíz se incluye el fichero `MLO`, que configura el hardware y ejecuta el fichero `u-boot.bin`. Se genera un script de configuración del arranque llamado `boot.scr`. Este script será ejecutado tras el fichero `u-boot.bin` y contiene los parámetros de arranque del fichero que deseamos ejecutar en el sistema empotrado.

2. Se genera una imagen de disco a la cuál se asigna en su primer sector el fichero `MLO`. Los siguientes ficheros son incluidos en direcciones de memoria específicas ya que usa *offsets* fijos. Las direcciones de memoria dependen de la placa. Para este método es necesario escribir las diferentes partes en sectores fijos de la MMC.

La estructura de la imagen de disco se puede observar en la Figura 4.4.

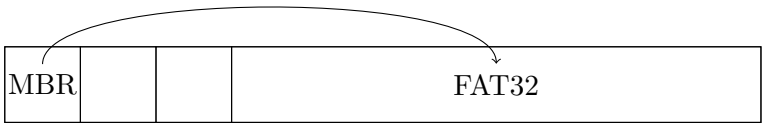


Figura 4.4: Estructura de la imagen de disco

Cuando se inicia la imagen de disco, la tabla de particiones busca la partición activa de tipo FAT. En este caso, la partición activa contiene en su directorio raíz el fichero MLO, que inicia el arranque del sistema. Un esquema del proceso de arranque se puede observar a continuación. El proceso puede se puede ver en la Figura 4.5.

[1] Esto es así para BeagleBoard. En otras placas cambiará.

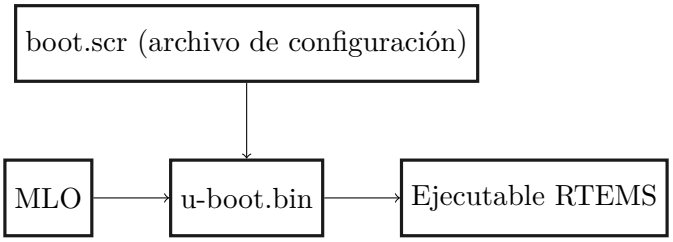


Figura 4.5: Proceso de arranque de la imagen

Sistema empotrado con software basado en cFE

Una vez generado el sistema empotrado que ejecuta un fichero de ejemplo de RTEMS, se procede a generar una imagen de disco que contenga el software de a bordo del nanosatélite. Para ello, se hace uso de los módulos implementados por los encargados del desarrollo del software de a bordo. Se ha hecho uso del *framework* cFE para esta tarea. cFE permite la compilación de los módulos para RTEMS. Para poder realizar la compilación con soporte para RTEMS, el fichero `Makefile` con ruta `cFE/apps/sch_lab/fsw/for_build` es modificado para incluir las librerías de RTEMS en el proceso de compilación de la aplicación. Para incluir las librerías de RTEMS en el fichero `Makefile` es necesario añadir en la línea `SHARED_LIB_LINK` del fichero lo siguiente: `-R../tst_lib/tst_lib.elf`.

Una vez modificado el fichero `Makefile` se procede a compilar la aplicación desarrollada, tal y como se puede observar en el Listado 4.7.

Listado 4.7: Comandos para compilar una aplicación de cFE para RTEMS

```
# Se establecen las variables de entorno
$ . ./setvars

#Se accede al directorio en el que se encuentra la aplicación que se desea compilar
$ make config
$ make
```

Los comandos para compilar una aplicación en cFE se encuentran en el Apéndice C, en la Sección C.5.

Cada módulo del ordenador de a bordo ha sido desarrollado por un estudiante diferente, por lo que existe una aplicación de cFE para cada módulo. Tras integrar las diferentes aplicaciones cFE, se ha procedido a la compilación del software. Se ha establecido como objetivo Linux (para la verificación Software-in-the-Loop), y RTEMS (para Hardware-in-the-Loop). De esta manera, se obtiene un ejecutable compilado y vinculado con el ejecutivo RTEMS. En el caso en el que los periféricos del nanosatélite sean simulados, en la compilación se incluyen las aplicaciones asociadas a dichas simulaciones.

Una vez compilado el software de a bordo (enlazado con el ejecutivo RTEMS), se procede a montar el sistema empotrado. El procedimiento a seguir es idéntico al explicado en la sección anterior. En este caso, se ha generado la tabla de particiones MBR con una partición activa FAT32, donde los ficheros `MLO` y `u-boot.bin` son idénticos a los usados en la subsección anterior. El ejecutable que se monta en la imagen de disco es el fichero que contiene el software de a bordo y que se ha compilado para RTEMS. En el script `boot.scr` se configuran los parámetros de arranque del ejecutable para que tras la ejecución del fichero `u-boot.bin` y se inicie el software de

a bordo. En la Figura 4.6 se puede observar el procedimiento de arranque del sistema empotrado generado.

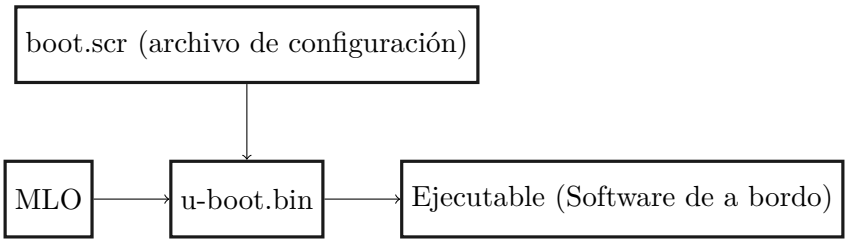


Figura 4.6: Proceso de arranque del software de a bordo

4.4. Entornos de prueba

Las pruebas realizadas sobre el producto se basarán en las fases de verificación *Software-in-the-loop* y *Hardware-in-the-loop*. Para ello, serán necesarios definir dos entornos de prueba.

El entorno software con el que se ha realizado la prueba es el siguiente:

- RTEMS 5
- Ubuntu 18.04
- gcc-arm-linux-gnueabi 7.3.0
- GNU make 4.1
- QEMU 2.3.50
- U-Boot SPL 2019.04

Las pruebas se realizaron sobre el siguiente hardware:

- System-on-a-Chip Texas Instruments OMAP3530 (emulado con QEMU para la fase *Software-in-the-loop*) sobre arquitectura Intel Core i7-2620M (x86_64).
- 8GB RAM

La estructura del entorno de pruebas se puede observar en las imágenes 4.7 y 4.8.

4.5. Criterio de aceptación de pruebas

A continuación se van a exponer los criterios que deben cumplir las pruebas definidas en la sección 4.6 para que sean aceptadas.

- La prueba se debe realizar en un entorno similar al descrito en la sección 4.4.

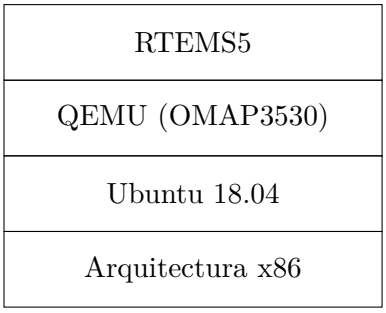


Figura 4.7: Entorno *Software-in-the-loop*

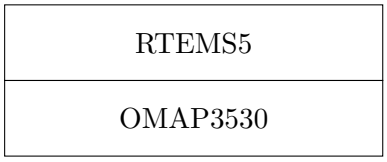


Figura 4.8: Entorno *Hardware-in-the-loop*

- La pruebas debe seguir el procedimiento descrito en su definición, que es especificado en los campos *entrada* y *descripción*.
- El resultado de la prueba tiene que coincidir con el especificado en el campo *salida*.

4.6. Definición de pruebas

Las pruebas se definen en una tabla con la estructura que se puede ver en la Figura 4.9, donde:

Identificador. Será CP-XX donde XX es el número de secuencia asignado al caso. La secuencia comienza en 01.

Descripción. Descripción del caso de prueba donde se especifica que se prueba.

Entrada. Entradas provistas para la realización de la prueba.

Salida. Resultado esperado de la ejecución de la prueba.

Entorno. Requisitos del entorno en el que se ejecuta la prueba.

Identificador

Descripción:

Entrada:

Salida:

Origen:

Entorno: n/a

Figura 4.9: Plantilla para la definición de los casos de prueba

Casos de prueba

CP-01

Descripción: Se prueba la ejecución de una imagen de disco en el emulador QEMU. La imagen de disco contiene el módulo de simulación del control térmico y el software de a bordo.

Entrada: RTEMS5 y on-board software

Salida: Se espera que:

- El sensor térmico comience a enviar valores simulados de temperatura.
- El software recibe los datos y los almacena en la tabla de telemetría.
- Si la temperatura es inferior al umbral mínimo, se activa el simulador del calentador.
- Si la temperatura es superior al umbral máximo, se desactiva el simulador del calentador.

Origen: RF-SR-03, RF-SR-04, RF-SR-05, RF-SR-06, RF-SR-07, RF-SR-08, RF-SR-14, RF-SR-15, RF-SR-16, RF-SR-17

Entorno:

- QEMU-linaro

CP-02

Descripción: Se prueba la ejecución de una imagen de disco en el hardware. La imagen de disco contiene el software de a bordo. Además, al GPIO se le conecta un termistor y un calentador para módulo del control térmico.

Entrada: RTEMS5 y on-board software

Salida: Se espera que:

- El sensor térmico comience a enviar valores de temperatura obtenidos del periférico.
- El software recibe los datos y los almacena en la tabla de telemetría.
- Si la temperatura es inferior al umbral mínimo, se activa el calentador.
- Si la temperatura es superior al umbral máximo, se desactiva el calentador.

Origen: RF-SR-03, RF-SR-04, RF-SR-05, RF-SR-06, RF-SR-07, RF-SR-08,RF-SR-13, RF-SR-14, RF-SR-15, RF-SR-16, RF-SR-17

Entorno:

- BeagleBoard-XM

CP-03

Descripción: Se prueba la ejecución de una imagen de disco en el emulador QEMU. La imagen de disco contiene el software de a bordo junto con el módulo de simulación del abastecimiento eléctrico.

Entrada: RTEMS5 y on-board software

Salida: Se espera que:

- El nanosatélite recibe la energía eléctrica suministrada por los paneles solares.
- El nivel de energía del nanosatélite aumenta.

Origen: RF-SR-11, RF-SR-12, RF-SR-14, RF-SR-15, RF-SR-16, RF-SR-17

Entorno:

- QEMU-linaro

CP-04

Descripción: Se prueba la ejecución de una imagen de disco en el hardware. La imagen de disco contiene el módulo del sensor solar. Además, se conecta al puerto GPIO correspondiente un grid de placas solares que conforma el sistema de abastecimiento eléctrico.

Entrada: RTEMS5 y on-board software

Salida: Se espera que:

- El nanosatélite recibe la energía eléctrica suministrada por los paneles solares.
- El nivel de energía del nanosatélite aumenta.

Origen: RF-SR-11, RF-SR-12,RF-SR-13, RF-SR-14, RF-SR-15, RF-SR-16, RF-SR-17

Entorno:

- BeagleBoard-XM

CP-05

Descripción:	Se prueba la ejecución de una imagen de disco en el emulador QEMU. La imagen de disco contiene el software de a bordo. Además, paralelamente se ejecuta el módulo de simulación del star tracker.
Entrada:	RTEMS5 y on-board software
Salida:	Se espera que: ▪ *Celestia* envíe el cuaternión correspondiente a la zona a la que apunta la cámara. ▪ El software transforme el cuaternión al eje de coordenadas, monitoriza su posición y almacena los datos calculados en la tabla de telemetrías.
Origen:	RF-SR-01, RF-SR-02
Entorno:	▪ QEMU-linaro

CP-06

Descripción:	Se prueba la ejecución de una imagen de disco en hardware. La imagen de disco contiene el software de a bordo. Además, paralelamente se ejecuta el módulo de simulación del star tracker conectado al puerto GPIO asociado.
Entrada:	RTEMS5 y on-board software
Salida:	Se espera que: ▪ *Celestia* envíe el cuaternión correspondiente a la zona a la que apunta la cámara. ▪ El software transforme el cuaternión al eje de coordenadas, monitoriza su posición y almacena los datos calculados en la tabla de telemetrías.
Origen:	RF-SR-01, RF-SR-02, RF-SR-13
Entorno:	▪ BeagleBoard-XM

CP-07

Descripción: Se prueba la ejecución de una imagen de disco en el emulador QEMU. La imagen de disco contiene el software de a bordo junto con la simulación del módulo de localización.

Entrada: RTEMS5 y on-board software

Salida: Se espera que:

- El módulo de localización envíe los valores de potencia eléctrica simulados.
- El software calcula la distancia respecto al sol a partir del valor recibido y almacena la posición en la tabla de telemetría.

Origen: RF-SR-09, RF-SR-10, RF-SR-14, RF-SR-15, RF-SR-16, RF-SR-17

Entorno:

- QEMU-linaro

CP-08

Descripción: Se prueba la ejecución de una imagen de disco en el hardware. La imagen de disco contiene el software de a bordo. Además, al GPIO se le conecta un grid de placas solares para módulo de la localización.

Entrada: RTEMS5 y on-board software

Salida: Se espera que:

- El módulo de localización envíe los valores de potencia eléctrica obtenidos del periférico.
- El software calcula la distancia respecto al sol a partir del valor recibido y almacena la posición en la tabla de telemetría.

Origen: RF-SR-09, RF-SR-10, RF-SR-13, RF-SR-14, RF-SR-15, RF-SR-16, RF-SR-17

Entorno:

- BeagleBoard-XM

4.7. Evaluación

La evaluación realizada a la ejecución de las pruebas definidas en la sección 4.6 se basa en la comprobación del éxito de las mismas. Se comprueba que el resultado de la ejecución de la prueba coincide con la salida esperada definida en el caso asociado.

Para este proyecto no se ha realizado un análisis de rendimiento puesto que al ser un sistema de tiempo real, lo único que se debe de verificar es que los plazos de tiempo se cumplan. Debido a que el planificador integrado en RTEMS asegura realizar el mayor esfuerzo (*best-effort*) para que los plazos se cumplan siempre, y asumiendo que el equipo encargado de la implementación de las tareas que se ejecutan sobre este sistema asegura que el tiempo de ejecución es inferior al tiempo de cómputo máximo, los cumplimientos de los plazos están garantizados.

4.8. Matriz de trazabilidad de pruebas

	RF-SR-01	RF-SR-02	RF-SR-03	RF-SR-04	RF-SR-05	RF-SR-06	RF-SR-07	RF-SR-08	RF-SR-09	RF-SR-10	RF-SR-11	RF-SR-12	RF-SR-13	RF-SR-14	RF-SR-15	RF-SR-16	RF-SR-17
CP-01			•	•	•	•	•	•						•	•	•	•
CP-02			•	•	•	•	•	•					•	•	•	•	•
CP-03											•	•		•	•	•	•
CP-04											•	•	•	•	•	•	•
CP-05	•	•															
CP-06	•	•											•				
CP-07									•	•				•	•	•	•
CP-08									•	•			•	•	•	•	•

Figura 4.10: Matriz de trazabilidad: casos de prueba - requisitos funcionales

Capítulo 5

Plan de proyecto

En este capítulo se incluye las tareas que han sido parte del desarrollo del proyecto y su planificación temporal. Además, se presenta el presupuesto requerido para llevar a cabo este proyecto.

5.1. Planificación del proyecto

El proyecto se encuentra dividido en el siguiente conjunto de tareas:

Propuesta del proyecto. Descripción del proyecto a desarrollar. Descrito por Javier López Gómez, Javier Fernández Muñoz y Jesús Carretero Pérez con la aprobación de la empresa SENER.

Análisis. Análisis de viabilidad y de requisitos. Incluye:

- Lectura de las referencias bibliográficas.
- Análisis de requisitos de usuario.
- Especificación de requisitos software.

Diseño del software a desarrollar en base a los requisitos especificados.

Implementación. Desarrollo de los objetivos del proyecto. Las fases realizadas para la implementación han sido las siguientes:

- Desarrollo de un sistema empotrado que contenga el sistema operativo Linux y que ejecute el conjunto de herramientas BusyBox. La ejecución del sistema se ha realizado en una máquina virtual.
- Desarrollo de un sistema empotrado que contenga los módulos del ordenador de a bordo, desarrollados en el *framework* cFE, compilados para RTEMS. El sistema empotrado es compatible con la arquitectura ARM.

Pruebas. Comprobación del correcto funcionamiento del software.

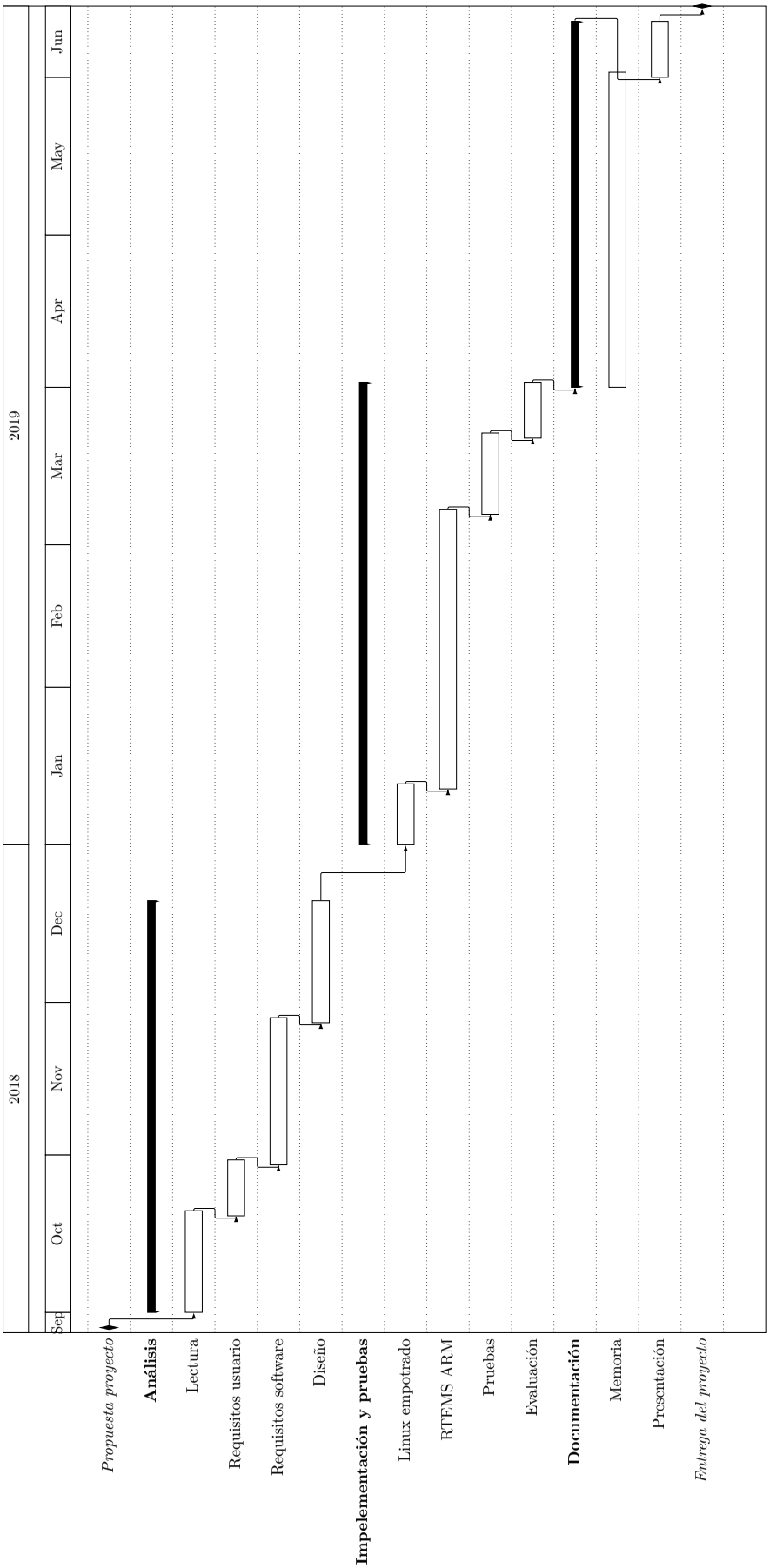
Evaluación del rendimiento de los sistemas desarrollados.

Documentación. Generar la documentación del proyecto:

- Memoria: éste documento.
- Presentación de diapositivas que acompaña a la memoria.

Este proyecto a tenido una duración de 8 meses y medio, siendo iniciado el 27 de septiembre de 2018 y finalizado el 11 de junio de 2019.

En la siguiente página se muestra la planificación temporal del proyecto realizado, para ello, se ha utilizado un diagrama de Gantt.


2018
2019
Sep
Oct
Nov
Dec
Jan
Feb
Mar
Apr
May
Jun
Propuesta proyecto
Análisis
Lectura
Requisitos usuario
Requisitos software
Diseño
Impelementación y pruebas
Linux empotrado
RTEMS ARM
Pruebas
Evaluación
Documentación
Memoria
Presentación
Entrega del proyecto

5.2. Presupuesto del proyecto

Todos los costes presentados en esta sección no tienen en cuenta el 21 % de I.V.A., salvo en el presupuesto final, donde se aplicará dicho impuesto.

Las tareas en las cual se desglosa el proyecto han sido descritas en la Sección 5.1. En la Tabla 5.1 puede observarse el total de horas invertidas en cada grupo de tareas.

Tarea	Horas
Análisis y Diseño	60 h
Implementación	110 h
Pruebas	40 h
Documentación	125 h
Total	335 h

Tabla 5.1: Desglose de horas por tarea

El proyecto ha sido realizado por varios ingeniero informáticos, con diferentes roles: un analista, un analista/programador y un programador. El sueldo medio por hora de un analista ronda los 45,00 €/hora, el sueldo medio por hora de un analista/programador de 28,00 €/hora y el de un programador de 17,00 €/hora. En los sueldos se encuentran incluidos los costes de seguridad social así como los costes indirectos relacionados con el personal (transporte, lugar de trabajo, etc). Al haberse invertido 335 h en el proyecto, el coste total del personal es de 8435,00 €, como se puede ver en la Tabla 5.2.

Cargo	Horas	Coste/Hora	Total
Analista	35 h	45,00 €	1575,00 €
Analista/Programador	160 h	28,00 €	4480,00 €
Programador	140 h	17,00 €	2380 €
Total	335 h	8435,00 €	

Tabla 5.2: Coste de personal

En el desarrollo del proyecto se ha hecho uso de software libre y gratuito. Sin embargo para ejecutar el sistema empotrado en la placa BeagleBoard-XM ha sido necesario comprar una placa. El gasto de este material se puede ver en la Tabla 5.3.

Material	Coste
BeagleBoardXM (modelo)	144,00 €
Total	144,00 €

Tabla 5.3: Coste del material empleado

A continuación, se muestran los costes derivados de la pérdida del valor del equipo en el cual se ha desarrollado el proyecto y que serán incluidos en el presupuesto. Además se incluyen los accesorios complementarios al equipo usados para facilitar su uso. Estos costes se pueden ver en la Tabla 5.4. La amortización aplicada al proyecto ha sido una amortización lineal, donde la fórmula aplicada para su cálculo ha sido:

$$Amortizacion = \frac{DuraciondelProyecto}{VidaUtil} * Precio * \%Uso$$

Recurso	Precio	Vida útil	% Uso	Repercutido
Portátil	250,00 €	48 meses	100 %	46,87 €
Ratón *Logitech* M310	19,99€	48 meses	100 %	3,70 €
Teclado *Logitech* K520	49,99€	48 meses	100 %	9,30 €
Pantalla *Samsung* S24D390Hl	212,99 €	48 meses	100 %	39,93€
Total				99,80 €

Tabla 5.4: Coste por amortizaciones

Además, se tiene que tener en cuenta el coste de todos los elementos que, a pesar de no repercutir directamente sobre el proyecto, suponen un gasto adicional durante el desarrollo del mismo. En el material de oficina se incluyen folios y bolígrafos. Para el gasto de electricidad se ha supuesto un coste de 0,31 €por KWh, con un consumo medio de 90W por hora. En cuanto a internet, se ha usado una tarifa de fibra óptica de 100Mb simétricos, que tiene un coste de 20,00 €al mes. En la Tabla 5.5 se detalla el coste que suponen estos elementos.

Descripción	Coste
Material	50,00 €
Internet	160,00 €
Electricidad	15,00 €
Total	225,00 €

Tabla 5.5: Costes indirectos

Tras haber considerado todos los elementos que suponen un gasto que debe ser imputado al proyecto, se procede a calcular el precio de venta de la solución que ha sido desarrollada. Se incluye un resumen de todos los gastos en la Tabla 5.6.

Para compensar imprevistos tales como la rotura de material o enfermedad del personal se aplicará un margen del 10 % sobre el precio. Por otro lado, el margen de beneficio que se va a aplicar al proyecto es de un 25 % sobre el precio con el margen de imprevistos.

El precio final de la solución, teniendo en cuenta el 21 % de I.V.A. es de 14 813,70 €, (**CATORCEMIL OCHOCIENTOS TRECE EUROS Y SETENTA CÉNTIMOS**).

Descripción	Coste
Personal	8435,00 €
Material	144,00 €
Amortizaciones	99,80 €
Indirectos	225,00 €
Margen de imprevistos (10 %)	890,38 €
Margen de beneficio (25 %)	2448,55 €
21 % I.V.A	2570,97 €
Total	14813,70 €

Tabla 5.6: Resumen del presupuesto

Capítulo 6

Entorno socio-económico y Marco legal

En este capítulo se exponen las principales regulaciones legales que se han tenido en cuenta en este proyecto. Además, se expone como el entorno socio-económico afecta de manera directa o indirecta al proyecto y el impacto económico del mismo.

6.1. Marco legal

Existen varias entidades reguladoras que se encargan de estandarizar proyectos del ámbito aeroespacial. A continuación se exponen algunas de ellas así como los diferentes estándares desarrollados.

CNAF

El Centro Nacional de Atribución de Frecuencias (CNAF) es la organización que se encarga de la asignación del espectro de radiofrecuencias en el territorio español. Especifica el uso de cada una de las bandas, que se encuentran dentro del rango de los 8,3 kHz y los 3000 GHz [48]. El CNAF se estructura en 3 columnas. En la primera se encuentran las asignaciones de frecuencias de acuerdo al reglamento de radiocomunicaciones establecido por la Unión Internacional de Telecomunicaciones (UIT). La segunda contiene las atribuciones de cada banda de frecuencia en territorio nacional si no coincide con la asignación de la primera columna. En la tercera columna se establece el uso del espectro que aplica cada banda de frecuencia. Hay cinco modalidades de uso: C (para uso común), E (para uso especial), P (para uso privado), R (para uso estatal) y M (para uso mixto entre el uso privado y el estatal).

CCSDS

El Comité Consultivo de Sistemas de Datos Espaciales (*Consultative Committee for Space Data Systems*) o CCSDS es una organización fundada por las principales agencias espaciales con el objetivo de desarrollar estándares de comunicaciones y sistemas de datos para misiones

espaciales [49]. La idea es que con estos estándares se mejore la interoperabilidad entre los diferentes Estados colaboradores al mismo tiempo que se reducen los riesgos, el tiempo de desarrollo y los costes de los proyectos. Los estándares desarrollados por esta organización se dividen en seis áreas [50]: Servicios de Internet en el espacio, servicios de gestión de la información y operaciones de las misiones, servicios de interfaz del ordenador de a bordo, ingeniería de sistemas, servicios de apoyo cruzado y servicios de enlace espacial.

ECSS

La Cooperación Europea para la Normalización del Espacio (*European Cooperation for Space Standardisation*) o ECSS es una iniciativa establecida para el desarrollo de un conjunto único de normas que deben ser aplicadas en cualquier actividad espacial en territorio perteneciente a la Unión Europea [51]. Cualquier empresa que colabore con la Agencia Espacial Europea (ESA) debe cumplir los estándares contemplados por la ECSS. Los estándares se dividen en dos ramas: los ECSS-E-40 se refieren al desarrollo del software mientras que los ECSS-Q-80 se refieren a la calidad del software. Entre los estándares desarrollados destacan [52]:

- **ECSS-E-40-01:** Estandarización del ordenador de a bordo.
- **ECSS-E-40-03:** Estandarización del ordenador de Tierra.
- **ECSS-E-40-04:** Estandarización del ciclo de vida del software.
- **ECSS-E-40-05:** Estandarización del proceso de verificación y validación del software.
- **ECSS-Q-80-01:** Directrices para la reutilización de software.
- **ECSS-Q-80-02:** Directrices para la evaluación y mejora de procesos de software.
- **ECSS-Q-80-03:** Directrices para la fiabilidad del software y métodos y técnicas de seguridad.

Tratados y principios de las Naciones Unidas sobre el derecho espacial

El derecho espacial es por la Oficina de Naciones Unidas para Asuntos del Espacio Exterior, que es la secretaría de la Subcomisión de Asuntos Jurídicos de la Comisión de las Naciones Unidas sobre la Utilización del Espacio con Fines Pacíficos (COPUOS). COPUOS es la única plataforma internacional para el desarrollo del derecho espacial. Desde su creación, ha establecido cinco instrumentos jurídicos y cinco conjuntos de principios jurídicos que rigen las actividades relacionadas con el espacio [53]. Entre ellos destacan:

El tratado sobre los principios que deben regir las actividades de los Estados en la exploración y utilización del espacio [54]. Este tratado se resumen en: la exploración y utilización del espacio se lleva a cabo en beneficio e interés de todos los países y es competencia de la humanidad. Los Estados no pondrán armas nucleares u otras armas de destrucción masiva en órbita ni en cuerpos celestes, ni las colocarán en el espacio de ninguna otra manera. La Luna y otros cuerpos celestes se utilizarán exclusivamente con fines pacíficos. Los Estados serán responsables de las actividades espaciales nacionales, tanto si se llevan a cabo mediante actividades gubernamentales como no gubernamentales. Los Estados serán responsables de los daños causados por sus objetos espaciales. Los Estados evitarán la contaminación nociva del espacio y de los cuerpos celestes.

El convenio sobre la responsabilidad internacional por daños causados por objetos espaciales [55]. Establece que el Estado será el responsable de pagar una indemnización por los daños causados por sus objetos espaciales en la superficie de la Tierra o a las aeronaves, así como por los daños causados por sus fallos en el espacio. El Convenio también establece los procedimientos para la resolución de reclamaciones de indemnización por daños y perjuicios.

El convenio sobre el registro de objetos lanzados al espacio [56]. Este convenio establece que el Estado debe proporcionar a las Naciones Unidas, la siguiente información relativa a cada objeto espacial:

- Nombre del Estado.
- Un identificador apropiado del objeto espacial o de su número de registro.
- Fecha y lugar del lanzamiento.
- Parámetros orbitales básicos, donde se encuentran incluidos: el periodo, la inclinación, el apogeo [1], el perigeo [2].
- Función general del objeto espacial.

Los principios jurídicos internacionales de esos cinco tratados controlan la no apropiación del espacio por un solo país, el control de armamentos, la libertad de exploración, la responsabilidad por los daños causados por los objetos espaciales, la seguridad y el rescate de naves espaciales y astronautas, la prevención de las injerencias perjudiciales en las actividades espaciales y el medio ambiente, la notificación y el registro de las actividades espaciales, la investigación científica y la explotación de los recursos naturales en el espacio. Los cinco conjuntos de principios jurídicos aprobados por la Asamblea General de las Naciones Unidas rigen la aplicación del derecho internacional y la promoción de la cooperación internacional en las actividades espaciales, la difusión y el intercambio de información entre Estados y la observación remota por satélite de la Tierra.

6.2. Entorno socio-económico

A partir de la década de 1960, como consecuencia de la guerra fría entre EEUU y la URSS, surgió la carrera espacial que tenía como objetivo invertir en misiones espaciales con el fin de obtener una posición de ventaja en el espacio. Debida a esta inversión se fueron sucediendo hitos que han terminado por convertir a la industria espacial como un elemento importante en nuestras vidas. Cada vez son más los servicios que se apoyan en infraestructuras espaciales para facilitar su distribución en la Tierra. Además, desde la década de 1960 se han realizando descubrimientos que han sido revolucionarios y han facilitado la vida en la Tierra. A continuación se exponen alguno de los factores que afectan a esta industria.

Factores económicos

Que un país tenga un número importante de infraestructuras en el espacio implica que dicho país es considerado una potencia mundial. Es por ello que cada vez más países invierten grandes cantidades de dinero en misiones espaciales.

[1] Altitud máxima sobre la superficie terrestre

[2] Altitud más baja sobre la superficie de la Tierra

Para la realización de proyectos espaciales han surgido diversas agencias que controlan dichas misiones, entre las que destacan la NASA y la ESA. Sin embargo, algunos países prefieren crear su propia agencia espacial. La mayoría de países de la zona europea pertenecen a la ESA, como es el caso de España.

Como demostración del aumento de la inversión en misiones espaciales, en el año 2015 el gasto de la NASA en misiones espaciales fue de 20.725 millones de dólares, mientras que en el año 1958 fue de 89 millones de dólares [57].

Factores tecnológicos

El uso de la tecnología en el espacio ha permitido que se hayan mejorado la prestación de los servicios existentes así como la aparición de nuevos. Algunos de estos servicios son:

Comunicación. La comunicación de dispositivos móviles se establece por medio de redes móviles. El uso de infraestructuras en el espacio ha permitido la mejora de las comunicaciones en la Tierra, incluso se ha conseguido que en lugares donde no hay cobertura se tenga acceso a la red a través de los satélites.

Localización. Con la aparición de los satélites se puede calcular la posición de un dispositivo con una alta precisión, simplemente con el uso de tres satélites con órbitas geoestacionarias. Entre otros usos, la localización puede ser utilizada para la navegación, misiones de rescate y seguimiento de mercancías.

6.3. Impacto económico

Este proyecto permite la integración de los módulos de un satélite independientemente del sistema operativo/arquitectura que se use. Esto facilita el proceso de integración del software de a bordo si se cambia un módulo del mismo o el hardware donde se integra. De esta manera, el dinero invertido en la integración puede verse reducido en las siguientes fases del proyecto.

El desarrollo del software de a bordo se realiza a través del *framework* cFE. Como se ha mencionado en la Sección 2.5 las aplicaciones desarrolladas con este *framework* pueden modificarse sin necesidad de reconstruir todo el sistema, ya que pueden ser compilarse, cargarse e iniciarse de manera independiente al sistema global. De esta manera, si se tuviera que realizar alguna modificación en uno de los módulos del ordenador de a bordo se ahorraría tiempo de recompilación del sistema. Por otro lado, cFE permite la reutilización de proyectos. Esto se debe a que cFE contiene un conjunto configurable de requisitos y código que permiten la adaptación de cFE a cada entorno. De esta manera, en un proyecto de colaboración como este, cada desarrollador puede probar la aplicación en un dispositivo y, tras comprobar que funciona correctamente, desplegar la aplicación en el sistema empotrado sin necesidad de tener que modificar el software, lo que ahorra tiempo en la verificación del sistema empotrado.

Por otro lado, se espera que las imágenes capturadas por el nanosatélite sirvan para el estudio de la radiación de la Tierra. Esto permitiría poder aumentar el conocimiento que se tiene actualmente sobre las características y el comportamiento de la Tierra.

Capítulo 7

Conclusiones y trabajos futuros

En este capítulo se incluyen las conclusiones del proyecto. En primer lugar, en la Sección 7.1 se incluye la verificación del cumplimiento de los diferentes objetivos definidos en la Sección 1.4. En la Sección 7.2 se exponen las conclusiones obtenidas del proyecto. En la Sección 7.3 se exponen las conclusiones personales respecto al proyecto. Por último, en la Sección 7.4 se describen algunas de las tareas que se podrían realizar si se desease continuar con el proyecto.

7.1. Objetivos

En esta sección se verifica el cumplimiento de los objetivos definidos en la Sección 1.4:

- Se ha conseguido integrar un sistema empotrado basado en Linux que ejecuta el conjunto de herramientas BusyBox. Esto ha permitido poder realizar un estudio de viabilidad del proyecto. De igual forma, ha permitido ahorrar tiempo en el desarrollo de las tareas que seguían a la consecución de este objetivo.
- Se ha desarrollado un sistema empotrado que contiene el sistema en tiempo real RTEMS compilado para la arquitectura ARM. Al arrancar el sistema, se ejecuta una aplicación compilada para RTEMS-ARM. Este proceso ha permitido verificar del correcto funcionamiento de RTEMS en un sistema empotrado.
- Se ha desarrollado un sistema empotrado que contiene el software de a bordo de un nanosatélite. El sistema se ejecuta sobre un RTOS con soporte para ARM. Para la verificación *Software-in-the-Loop*, los periféricos del nanosatélite han sido simulados.
- Todo el software desarrollado ha cumplido las fases de verificación *Software-in-the-Loop* y *Hardware-in-the-Loop*.

7.2. Conclusiones del proyecto

Al pertenecer este Trabajo de Fin de Grado a una Cátedra, se ha necesitado de un alto grado de coordinación entre los distintos participantes de la misma. Para el desarrollo del proyecto, se han realizado reuniones semanales entre los integrantes de la Cátedra y con SENER, de tal

manera que se realizaba un seguimiento de los diferentes progresos que se iban consiguiendo. Durante estas reuniones, se han ido determinando detalles del proyecto como la arquitectura del nanosatélite y los requisitos del proyecto. Al haber sido el primer año de la Cátedra. los primeros meses fueron destinados a la investigación, análisis y diseño del proyecto. Es por ello que la implementación del proyecto no comenzó hasta comienzos del presente año.

Este proyecto ha permitido conocer el proceso de arranque de un sistema empotrado, y como debe de ser configurado en función de la arquitectura. Para el arranque del sistema se ha hecho uso del gestor de arranque U-Boot, que tiene soporte para múltiples arquitecturas y permite la configuración personalizada del sistema. Sobre el sistema empotrado se ejecuta el sistema operativo de tiempo real RTEMS que, al igual que U-Boot, tiene soporte para múltiples arquitecturas. Para el software de a bordo, el equipo encargado del desarrollo de esta tarea ha hecho uso del *framework* cFE. Los módulos del software se han desarrollado por separado, y una vez completados se han integrado para formar el software de a bordo. El resultado de la integración se ha compilado para RTEMS, que genera el fichero que es ejecutado al arrancar el sistema empotrado del nanosatélite. El software de a bordo es el encargado de monitorizar los valores enviados por los periféricos.

El software desarrollado por el proyecto ha sido verificado por medio de las fases *Software-in-the-Loop* y *Hardware-in-the-Loop*. Para la verificación *Software-in-the-Loop* se ha hecho uso del software QEMU-Linaro, que permite la emulación de hardware. De esta manera, se puede comprobar el funcionamiento del sistema empotrado sin necesidad de usar hardware real. Además, para esta fase, los periféricos han sido simulados a través de aplicaciones implementadas, usando cFE, por el equipo de desarrollo del software de a bordo. Para la verificación *Hardware-in-the-Loop*, se ha utilizado una placa de desarrollo BeagleBoard-XM. El sistema empotrado era cargado en el *System-on-a-Chip* del hardware. Los periféricos del nanosatélite se han conectado al hardware por medio de los puertos GPIO habilitados.

Durante el desarrollo del proyecto se han encontrado diversos problemas, sobre todo a la hora de realizar la fase de verificación *Software-in-the-Loop*, ya que para emular BeagleBoard-XM y que funcionase, era necesario configurar el proceso de arranque del hardware a través del emulador. Sin embargo, una vez realizada la verificación *Software-in-the-Loop*, la verificación *Hardware-in-the-Loop* se ha logrado sin demasiados inconvenientes.

7.3. Conclusiones personales

El Trabajo de Fin de Grado ha supuesto uno de mis mayores retos durante el grado. Para su desarrollo ha sido necesario un gran esfuerzo en cuanto a implicación y compromiso. La Cátedra ha supuesto una magnífica oportunidad, ya que ha permitido aplicar alguno de los conocimientos adquiridos durante el grado en un proyecto espacial educativo que algún día se convertirá en realidad.

El proyecto ha permitido afianzar los conocimientos adquiridos durante la asignatura *Sistemas en Tiempo Real*, que han sido necesarios para la elección del sistema operativo de tiempo real así como para el desarrollo del sistema empotrado. Además, se ha hecho uso de los conocimientos de ingeniería del software adquiridos durante la carrera. Para las tareas de ingeniería del software se ha usado un paquete de LaTeX2e cedido por Javier López Gómez [1]. Este paquete permite la automatización de proceso de generación de las tablas de requisitos, casos de uso, etc.

[1] Este paquete es accesible a través de la siguiente url `https://github.com/jalopezg-uc3m/SRS-latex-uc3m`.

De igual manera se ha aprendido a configurar un emulador en función de las especificaciones del sistema empotrado. Además, se ha conocido el proceso de configuración del gestor de arranque U-Boot para una plataforma ARM.

Por último, para la realización del presente documentos se ha aprendido LaTeX2e, mientras que para la generación de algunas de las figuras incluidas en el documento se ha aprendido *TikZ*.

7.4. Trabajos futuros

El proyecto tiene algunas mejoras que se podrían añadir. A continuación se exponen algunas de las más importantes para continuar con el proyecto.

Integrar el proceso de generación del sistema empotrado en Eclipse. Para facilitar el proceso de creación de la imagen de disco, se podría automatizar el proceso a través de la herramienta Eclipse. De esta manera no sería necesario el uso de una máquina virtual.

Soporte en Eclipse para la depuración mediante la interfaz JTAG. A medida que el código sea más complejo, será deseable depurar la ejecución directamente sobre el hardware. Para ello, la placa elegida dispone de una interfaz JTAG. Con el fin de facilitar su uso, la depuración hardware-in-the-loop podría ser integrada en Eclipse.

Generar el sistema empotrado para el hardware final. Una vez se haya decidido en la cátedra el hardware que incluirá el nanosatélite se debe repetir el proceso de creación del sistema empotrado pero con soporte para el hardware elegido.

Apéndice A

Summary

A.1. Introduction

This project is part of a joint chair between UC3M and SENER. SENER is a Spanish engineering company working in sectors such as energy and aerospace. This multidisciplinary chair, which brings together students of computer engineering, aerospace..., aims to build a nanosatellite with the collaboration of SENER. It is estimated that the launch date of the nanosatellite will be approximately in the year 2022.

A.1.1. Motivation

The architecture of nanosatellites arises due to the emergence of the philosophy *"New space"*. This has changed the space landscape. Its objective is the colonization of space through small satellites that are cheaper and faster to manufacture compared to classic satellites. This current has allowed companies that do not currently own satellites in space, and believe that it is necessary to have an infrastructure in space in order to be able to provide their services, to be able to enter the space business. Above all, these companies are start-ups, newly created companies which, in order to enter a competitive market, need to have such an infrastructure in space.

As nanosatellites are capable of providing the same service as a satellite and are less expensive, the number of companies applying this philosophy is increasing. This means that the technology applied by nanosatellites is increasingly used in the space landscape, so this project uses technology that is currently on the rise.

A.1.2. Embedded system

An embedded system is a controller that is managed by a RTOS whose design is based on performing specific tasks with real-time restrictions. A RTOS (Real Time Operating System) is an operating system that has been developed for real-time applications and must meet temporary requirements for proper operation. Most of the components of an embedded system are located on the motherboard, such as the graphics card or modem. Most modern embedded systems are microcontrollers that allow the system to be reduced in size. A microcontroller is a

programmable integrated circuit, capable of executing commands stored in its memory. Designers are continuously studying the possibility of optimizing the system to reduce size and cost and improve the performance of performing tasks. Currently, most embedded systems are mass produced to increase profits.

The main manufacturers of embedded systems are Intel, Motorola, AMD and STMicroelectronics. Some of them are able to integrate the microprocessor and the control elements of the input and output elements in a single chip to meet the specifications of the systems that have been described above. This technique is applied to systems that do not require much power as the processing capacity is lower than that of traditional embedded systems.

The operating systems that are necessary for an embedded system to function and run programs are usually special systems for embedded systems. They are systems that have few memory requirements, that include the possibility of containing applications in real time and that allow incorporating only the necessary modules for a specific embedded system. The best known today are FreeRTOS, RTEMS and VxWorks.

As mentioned above, an example of the constant improvement in technology related to satellites are the embedded systems for controlling them. This can also be applied to a *CubeSat*[4]. A CubeSat is a smaller satellite whose maximum weight does not exceed 1.3 kilograms. Its standard size is 10x10x10 centimetres which makes it the shape of a cube. The *CubeSats* can be combined to form a nanosatellite. A nanosatellite is any satellite weighing less than 10 kilograms. The use of nanosatellites is increasing, as they can perform the same functions as a traditional satellite, resulting in more and more artificial objects in space.

Nowadays, the use of nanosatellite is increasing to perform experiments with new technology that will appear as is the case of graphite transistors. The onboard system of a nanosatellite needs to have a specific structure similar to that of Figure A.1. In this figure, it can be seen how the applications and services of the nanosatellite run on the operating system, regardless of which operating system is used.

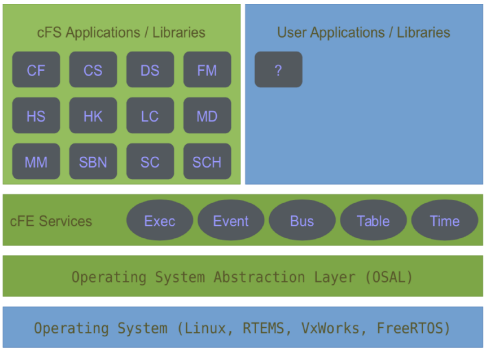


Figura A.1: Example of the structure of the On-Board Software [1]

A.1.3. Hardware Platform

A platform, in the field of computing, refers to the system that serves as the basis for hardware and software modules to function correctly as a whole. When defining a platform it is necessary to establish the type of architecture and the operating system. The following are some examples of hardware architectures. It is important to know the different architectures and processors for the implementation process of the platform since the development of the embedded platform will depend on the architecture we are using. In addition, its characteristics

must be known in order to choose the one that best suits the needs of the project.

In this project, it has been decided to use ARM architecture, as it is the architecture that best suits the needs of the project. ARM has a wide range of microcontrollers that support a RTOS and their consumption is relatively low. This is important considering that the only source of electrical supply for the nanosatellite will be solar energy, and the nanosatellite will not be able to complete its mission if the embedded platform does not have enough energy, so the less it consumes the less probability of running out of energy there will be. Going into more detail, ARM's CPU architecture uses microarchitecture techniques to support a wide range of ARM performance points. The CPU architecture defines the set of basic instructions as well as the exception and memory models on which the operating system (and hypervisor if any) is based. The microarchitecture defines processor power, performance and area by determining the length of the *pipeline*, cache levels, etc. Originally, the CPU architecture was based on the principles of the RISC computer and incorporated a load and storage architecture, in which data processing worked only with the content of the registry, and not with the content of the memory. It included simple addressing modes, in which all loading and storage addresses were determined from the contents of the record and the instruction fields. Some types of hardware platforms are going to be shown below.

BeagleBoard

One type of ARM hardware is BeagleBoard [13]. BeagleBoard is a type of free, low-power ARM hardware board. It is produced by *Texas Instruments* with the collaboration of *Newark element14* and *DigiKey*. Texas Instruments used this type of board to demonstrate the capabilities of the *OMAP3530*. OMAP3530 devices are based on the architecture *OMAP3* [14]. The OMAP3 architecture is designed to provide sufficient video, image and graphics processes to support real-time video, video conferencing and high-resolution imaging. OMAP3530 devices support high-level operating systems such as Linux, Windows and Android. In addition, ARM boards allow the execution of real-time systems such as RTEMS. You can find the following types of BeagleBoard boardsBB-TYPES:

BeagleBoard Original. It's a development plate launched in October 2011. Some of its characteristics are: AM3358 ARM Cortex-A8 processor, 256 MB RAM memory and processor clock running at 720 MHz. The Original BeagleBoard includes JTAG connections with hardware debugging, so there is no need for a JTAG emulator. It is also known as BeagleBone. The BeagleBone plate is priced at 84.99 €, according to the company *DigiKey* [16].

BeagleBone Black. It's a low-cost development platform for developers and hobbyists. It was released in April 2013. Some of its characteristics are [17]: AM3358 ARM Cortex-A8 processor, 512 MB RAM memory and processor clock running at 1 GHz. The BeagleBone Black includes a 2GB flash memory. In May 2014 was released a version, BeagleBone Black C, which increased the size of the flash memory to 4GB. The BeagleBone Black plate has a price of 54.99 €, according to the company *DigiKey* [18].

BeagleBone Blue. It's an all-in-one Linux-based board which integrates into a single small board. Some of its characteristics are [19]: AM335x ARM Cortex-A8 processor, 512 MB RAM memory and processor clock running at 1 GHz. The hardware includes a battery for self-powering. It has a market price of 82.99 €, according to the company *DigiKey* [20].

BeagleBoard-X15. This is the most sophisticated and high-performance board in the BeagleBoard family. Some of its characteristics are BB-X15: AM5728 ARM Cortex-A15 processor, 2 GB RAM memory and processor clock running at 1.5 GHz. BeagleBoard-X15 includes a high-speed interface for every connectivity need. Its price is around 215,99 €, according to the website *Arrow* [22].

BeagleBoard-XM. It's a modification of the Original BeagleBoard. It was released in August 2010. Some of its characteristics are [23]: AM37x ARM Cortex-A8 processor, 512 MB RAM memory and processor clock running at 1GHz. BeagleBoard-xM eliminates NAND, this makes the operating system and data to be stored on a microSD card. Its price in the market is around 144,00 €, according to the company *DigiKey*. [24].

A.1.4. Real-Time Operating System

A real-time operating system (RTOS) is an operating system that has been developed for real-time applications [3]. These systems must have temporary requirements for operations. There are two main groups:

Critical systems. Are those that must guarantee the completion of all tasks within their corresponding response time. If a deadline fails, the system may fail fatally. Also known as *hard RTOS*. Examples of critical RTOS are the antilock breaking system (ABS) of a vehicle or a pacemaker.

Non-critical systems. Are those that can afford to miss a response deadline sporadically. Within this group, two types of RTOS can be differentiated: pure non-critical (or soft RTOS, which tries to meet deadlines but if not, the system continues to work, although degraded) and firm non-critical (or firm RTOS, where results obtained after the deadline are useless but the system does not collapse. Examples of non-critical RTOS are video processing or air-conditioning.

Due to the availability of several RTOS, the following paragraphs will briefly describe some of them to show the difference between them.

VxWorks. It's an RTOS developed by Wind River Systems. [26]. It is usually used in embedded systems that require a quick response of about milliseconds to the interruptions that arrive. Its structure is based on a micro-kernel of an approximate size of 20KB, a working environment and a set of applications. Normally, you work with it via a remote connection. An example of the use of VxWorks can be found in the robots *Rover* used for geological purposes by NASA [27]. The RTOS controlled the transport unit of the *Rover* and provided communications to the ground.

FreeRTOS. This is an embedded deviceRTOS licensed under the MITFreeRTOS license. FreeRTOS is designed to be small and simple. The kernel consists of only three files written in C language so that the code is readable, easy to transport and maintain. However, it includes some functions written in assembly language such as those included in architectural planning routines. FreeRTOS does not support advanced memory management. It also supports the most popular socket and transport layer libraries, such as *wolfSSL*.

RTEMS. RTEMS is a RTOS developed as free software and designed for embedded systems that require a quick response, some security and stability. It supports programming interfaces of standard applications such as *POSIX*. Its use can be found in space flight,

medicine, networks, etc. RTEMS is the selected RTOS for the development of the project. The structure of RTEMS is based on micro-kernel, a development environment and several applications designed to make the device work properly. Normally, you work with it through a remote connection, where you develop the software that is going to be loaded into the final device. Among other features are: It is available for architectures such as *ARM, PowerPC, Intel or SPARC*, RTEMS has a data organization structure based on the standard defined by *POSIX*, where the supported file systems include: a file system type file allocation table (FAT), a proprietary data system (*RTEMS File System*), and the network protocol NFS, It supports UDP and TCP internet protocols, The kernel has multitasking capability with the dynamic memory location and It is compatible with USB communication protocols, SD/MMC cards, among others.

A.1.5. cFE

Core Flight Executive (cFE)- see Figure A.1- is a software developed by NASA (*National Aeronautics and Space Administration*) of free access that serves as an environment to develop applications and execute themCFE. cFE provides facilities for messaging, event processing, etc. cFE defines an Application Programming Interface (API) for each service that shall be used for application development.

In regard to messaging, it provides a publisher/subscriber system that allows applications to easily communicate with each other. Therefore, applications might subscribe to different topics —*pipes* in cFE terms— at runtime, allowing modifications to be made to the system more easily. Additionally, cFE allows new applications to be compiled, loaded and started without rebuilding the entire system. It also allows applications to be tested on a different configuration and, after checking it is working correctly, to deploy the application on an embedded system without modification.

As can be seen in Figure A.1, the system layers include an operating system abstraction layer (*OSAL*), a flight execution layer (cFE) and an application layer (where the different nanosatellite applications are located). The cFE layer is executed on the *OSAL* layer. *OSAL* is available as open source.

Therefore, cFE and *OSAL* provide the basis for building an on-board software that can run on top of an RTOS.

A.1.6. U-boot

As the project is based on integrating a disk image into an embedded system, it must have a boot system that allows the hardware to be initialized to run the embedded system. To do this, a less heavy boot system than the Linux *GRUB* must be used. In this project it has been decided to use *U-Boot* as this boot system is found on most embedded systems and allows you to package the kernel instructions to boot the device operating system. *U-Boot* is a free software system that supports several types of architectures, including ARM, MIPS and x86 [33].

The system ROM or BIOS loads U-Boot from a compatible boot device, such as an SD card. U-Boot's boot can be divided into two stages [34]: the platform loads an SPL (*Secondary Program Loader*) which performs the initial hardware configuration and loads the full version of U-Boot. Unlike other boot systems that automatically select kernel memory locations and

other boot data, U-Boot requires that memory addresses be specified as destinations for copying data.

U-Boot does not need to read a file system for the kernel to use as a root file system or initial RAM disk. It provides a specific parameter to the kernel and copies the data to memory. However, U-Boot can also read a file system. This way, instead of the data that U-Boot loads being stored in a fixed location on the storage device, U-Boot can read the file system to find and load the kernel, device tree, etc., through the path. Among the file systems supported by U-Boot are FAT 32 and ext4.

Development enviroment

The following decisions have been taken regarding the development environment of the project:

- The architecture of the hardware platform is ARM. Your choice is because you have a wide range of microcontrollers that support an RTOS and the consumption of them is relatively low. In addition, it is an architecture commonly used in space missions since it offers high performance in embedded systems and for real-time systems.
- The framework used for the development of the simulation of the nanosatellite modules is cFE. This is because it is an open source framework that has been used in multiple missions. In addition, it allows the different modules of the nanosatellite to be compiled, loaded and started without having to rebuild the whole system. cFE allows each module to be tested independently and, after checking its correct operation, to integrate all the modules in an embedded system without modifying the software.
- The RTOS on which the modules of the nanosatellite run is RTEMS. RTEMS offers full support for the framework. In addition, it is a *open source* system with support for most hardware architectures, including ARM. RTEMS is designed for embedded systems that require a quick response, some security and stability.
- The boot loader for the disk image is U-Boot. It has been decided to use U-Boot as it is the boot system used in most embedded systems because it is a lightweight system and allows you to package the kernel instructions to boot the device operating system. In addition, U-Boot supports the ARM architecture.

A.2. Architecture

As the chair is in the initial phase, all the details of the real mission of the nanosatellite have not yet been defined, only it is known that one of its functions will be to take photographs of the Earth. In order to carry out this first phase, a simple mission has been defined. This mission consists of the onboard software carrying out periodic processing of the values obtained from the different sensors that make up the nanosatellite. After collecting these values, it will have to act in function of the obtained value and it will have to send the telemetries to the earth computer so that it monitors these values. In addition, the onboard computer contains a failure control mechanism that monitors the values provided by the sensors. Sockets will be used for communication between the onboard and ground computers. In Figure A.2 you can see a system schema where you can differentiate the different modules and components of the

mission. In this project only the hardware simulation modules, onboard software and hardware (located inside the box) will be taken into account, as it consists of integrating these modules into an embedded system and verifying their operation through the phases *software-in-the-loop* and *hardware-in-the-loop*. Below is a brief explanation of the different elements that make up each module.

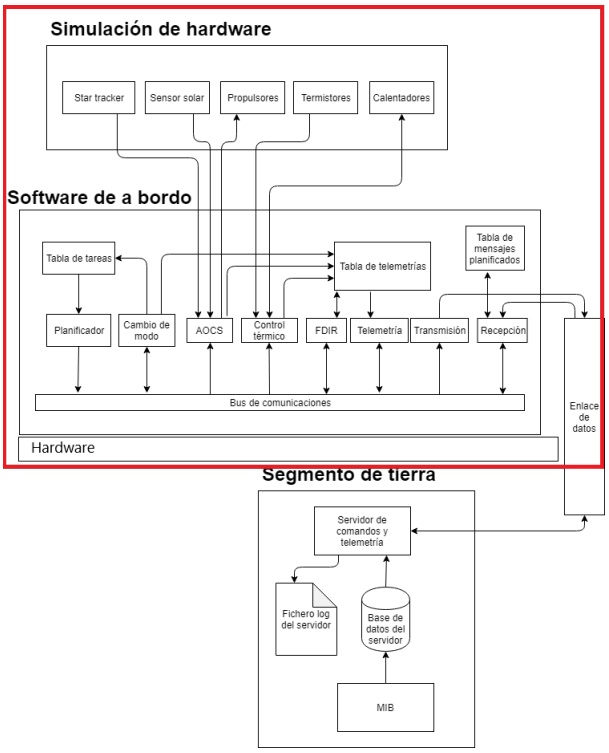


Figura A.2: Scheme of the architecture of the global project (the image has been ceded by the student Pablo Andreu Sedeño)

The following elements must be taken into account in the first module:

Star tracker. Its objective is to determine the altitude of the nanosatellite. It consists of a camera integrated into the backwards. It returns a quaternion [1] with respect to an inertial axis. The simulation of this module is done through images obtained through the tool *Celestia*.

Solar sensor. Sensor that is formed by several panels that capture the electrical potential received from the sun. In this way, it is possible to calculate the position of the sun with respect to the nanosatellite.

Propellants. Actuators that are activated when you want to change the altitude of the nanosatellite. This actuator will be simulated.

Thermistors. Sensors that will take care of obtaining the temperature value. The sensor will be simulated through a sine wave between a range of maximum and minimum temperature values.

Heaters. Actuators that are activated as a function of temperature. They allow increasing the temperature of the zone of the nanosatellite in which they are. There is a heater for each thermistor. Their operation will be simulated, when they are turned on the temperature value of the thermistor sensor associated with the activated heater will increase.

[1] The quaternion represents the orientation and rotation of the nanosatellite in three dimensions.

The second module consists of the onboard software. It will monitor the values returned by the elements of the first module. Among other elements, it contains the thermal control, the task planner and the telemetry table.

The module *Hardware* takes care of the execution of the onboard software together with the hardware simulation. Therefore, the module contains the System-on-a-Chip in which the software will be loaded to be executed. This module covers both the verification *Software-in-the-loop* and the verification *Hardware-in-the-loop*.

A.3. Targets

The following is a list of the objectives to be achieved in the development of this project. Some of the objectives include secondary objectives that are relevant to development.

Integrating an embedded system incorporating an RTOS . A disk image must be generated that contains an embedded system with a RTOS. This image must be integrated into an embedded system together with the system configuration and boot files. The execution of this image must be tested in a hardware simulator as well as in real hardware. This objective contains several secondary objectives necessary for the study of the viability of the development.

Testing with an embedded Linux-based system. A disk image should be generated for an embedded system based on *Linux*. This image is a test mode containing BusyBox.

Generate a bootable image of a RTOS for architecture ARM. A disk image containing an embedded system containing the system in real time RTEMS compiled for the architecture ARM must be generated. When you run that machine, it should show the execution of a compiled application in RTEMS from ARM.

A.4. Implementation

The disk image contains the RTEMS executable along with the system boot system (U-Boot). The U-Boot's boot can be divided into two phases: first, the SPL is loaded, which performs the initial configuration of the hardware, and the full version of U-Boot is loaded. Therefore, two files are required to follow this procedure. The SPL is a file named `MLO` [2] and the file with the full version of U-Boot is a file named u-boot.bin. Both files are generated during U-Boot compilation and are located in its root directory. The last main element for the image to boot is the executable compiled and linked to the RTEMS executive.

The image will be tested on Texas Instruents' OMAP3 hardware, either simulated or real. OMAP3's System-on-a-Chip has two boot procedures:

1. A MBR (*Master Boot Record*) partition table is generated with an active partition in FAT12/16/32. In its root directory is included the MLO file, which configures the hardware and executes the file u-boot.bin. A boot configuration script called *boot.scr* is generated.

[2] That's the way it is for BeagleBoard. On other hardware, it will change.

This script will be executed after the u-boot.bin file and contains the boot parameters of the file we want to run on the embedded system.

2. A disk image is generated to which the MLO file is assigned in its first sector. The following files are included in specific memory addresses as they use fixed *offsets*. The memory's addresses depend on the board. For this method, it is necessary to write the different parts in fixed sectors of the MMC.

The structure of the disk image can be seen in Figure A.3.

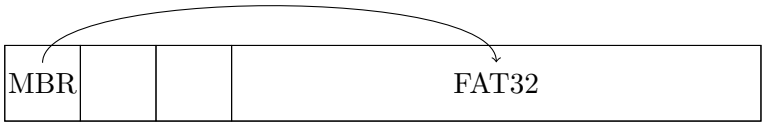


Figura A.3: Disk image structure

When the disk image starts, the partition table searches for the active FAT partition. In this case, the active partition contains in its root directory the MLO file, which starts the system boot. An outline of the boot process can be seen below. The process can be seen in Figure A.4.
.

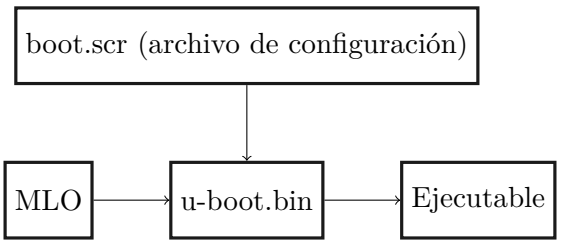


Figura A.4: Image boot process

Once the embedded system that executes an RTEMS file is generated, a disk image containing the software on board the nanosatellite is generated. To do this, use is made of the modules implemented by those in charge of developing the onboard software. Use has been made of the *framework* cFE to develop the onboard software. cFE allows the compilation of modules for RTEMS. To do this, the file `Makefile` located in path `cFE/apps/sch_lab/fsw/for_build` is modified to include the RTEMS libraries in the application compilation process. Each module of the onboard computer has been developed by a different student, so there is a cFE application for each module. After integrating and communicating the modules, the integration has been compiled for RTEMS. In this way, an executable compiled and linked to the RTEMS executive is obtained.

Once the executable is obtained, the embedded system is assembled. The procedure to follow is identical to the one explained in the preceding paragraphs.

A.5. Results

The tests performed on the implementation focus on verifying the embedded system. To do this, software-in-the-loop and hardware-in-the-loop verifications are performed. The tests

consist of checking the behaviour of the system by executing a disk image containing the onboard software and nanosatellite modules (temperature control, solar sensor, etc.). The nanosatellite peripherals are simulated in the software-in-the-loop phase and connected to the hardware through GPIO ports in the hardware-in-the-loop phase. To perform these verification phases, two test environments will need to be defined.

The software environment in which the test was performed is as follows: RTEMS 5, Ubuntu 18.04, gcc-arm-linux-gnueabi 7.3.0, GNU make 4.1, QEMU 2.3.50 and U-Boot SPL 2019.04.

The tests were performed on the following hardware: OMAP3530 processor, 8GB RAM and 250 GB ST320LT007-9ZV142 hard disk.

The structure of the test environment can be seen in the images A.5 and A.6.

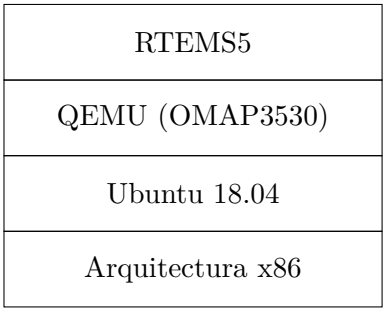


Figura A.5: *Software-in-the-loop* enviroment

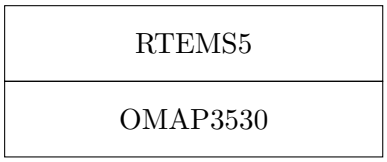


Figura A.6: *Hardware-in-the-loop* Enviroment

The evaluation of the execution of the tests defined in section 4.6 is based on the verification of their success. It is verified that the result of the execution of the test coincides with the expected output defined in the associated case.

For this project, performance analysis has not been carried out, since being a real-time system, the only thing to verify is that the time limits are met. Because the planner integrated into RTEMS ensures that best-effort is searched to ensure that deadlines are always met, and assuming that the team responsible for implementing the tasks that are executed on this system ensures that the execution time is less than the maximum computation time, compliance with deadlines are guaranteed.

A.6. Conclusions and future research lines

A.6.1. Product conclusions

- To study the viability of the project, a Linux embedded system that runs the BusyBox toolkit has been developed.
- The product integrates the software of the nanosatellite onboard computer into an embedded system along with the sensors of the same.
- The product has two phases of verification: *Software-in-the-loop* and *Hardware-in-the-loop*.
- The embedded system includes the U-Boot's boot loader as it allows custom configurations and has support for various hardware platforms.

- Booting via U-Boot is done through an SPL (*Secondary Program Loader*), which is loaded by the platform, and which performs the initial hardware configuration and loads the full version of U-Boot.
- The onboard software is run on the real-time system RTEMS as it has support for different architectures and is open-source.
- The compilation and installation of RTEMS and U-Boot depend on the architecture on which the embedded system is going to run.
- For the phase *Software-in-the-loop* the QEMU emulator is used. For BeagleBoard-XM machines it is necessary to install the unofficial QEMU-Linaro version, which provides support for the machine.

A.6.2. Personal conclusions

- Knowledge related to real-time operating systems has been acquired.
- Software engineering knowledge acquired during the degree has been applied in the project. For the engineering tasks of the software has been used a package LaTeX2e provided by Javier López Gómez [3].
- Insights into embedded boot-loaders have been gained..
- We have learned to configure an emulator according to the specifications of the embedded system.
- We have learned to integrate an application into an embedded system.
- The U-Boot's boot process for an ARM platform has been learned.

A.6.3. Future research lines

The project has some improvements that could be added. Here are some of the most important ones to continue with the project.

Integrate the embedded system generation process in Eclipse. To facilitate the process of creating the disk image, the process could be automated through the Eclipse tool. Using this process, it would not be necessary to use a virtual machine.

Support in Eclipse for debugging via the JTAG interface. As the code becomes more complex, it will be desirable to debug the execution directly on the hardware. For this, the chosen board has a JTAG interface. For ease of use, hardware-in-the-loop debugging could be integrated into Eclipse.

Generate the embedded system for the final hardware. Once the hardware that will include the nanosatellite has been decided in the chair, the process of creating the embedded system must be repeated but with support for the chosen hardware.

[3]This package is accessible through the following url `https://github.com/jalopezg-uc3m/SRS-latex-uc3m`.

Apéndice B

Glosario

ARM. Arquitectura con un conjunto de instrucciones reducidas que fue diseñada inicialmente por *Sun Microsystems*.

Beagleboard. Tipo de placa ARM de bajo consumo, producida por *Texas Instruments* con la colaboración de *Newark element14* y *DigiKey*.

cFE. *Core Flight Executive*. Software desarrollado por la NASA de acceso libre que sirve de entorno para desarrollar aplicaciones y ejecutarlas.

CubeSat. Satélite de menor tamaño cuyo peso máximo no sobrepasa los 1.3 kilogramos. Su tamaño estándar es de 10x10x10 centímetros lo que hace que tenga la forma de un cubo.

Firmware. Programa que determina la lógica, de más bajo nivel, que controla el hardware de un dispositivo.

Microcontrolador. Circuito integrado programable, capaz de ejecutar las órdenes almacenadas en su memoria.

Nanosatélite. Cualquier satélite que pese menos de 10 kilogramos que surge de la unión de varios *CubeSats*.

OMAP3. Arquitectura diseñada para proporcionar procesos de vídeo, imagen y gráficos suficientes para soportar vídeo en tiempo real, videoconferencia e imagen de alta resolución. Tipo de arquitectura de los procesadores de las placas Beagleboard, que permiten la ejecución de RTOS.

Qemu-linaro. Extensión no oficial del emulador QEMU que incluye soporte para la simulación de hardware de BeagleBoard.

RTEMS. *Real-Time Executive for Multiprocessor Systems*. Tipo de RTOS desarrollado como software libre y diseñado para sistemas empotrados.

RTOS. *Real Time Operating System*. Sistema operativo que ha sido desarrollado para aplicaciones de tiempo real , y que debe cumplir unos requisitos temporales para que el funcionamiento sea el adecuado.

SENER. Empresa española del ámbito de la ingeniería que trabaja en sectores como el de las energías y el aeroespacial.

Sistema empotrado. Un sistema empotrado es un controlador que es manejado por un RTOS cuyo diseño se basa en realizar unas tareas específicas con restricciones de tiempo real.

SPL. *Secondary Program Loader*. Realiza la configuración inicial del hardware y carga la versión completa de U-Boot.

System-on-a-Chip. Circuito integrado que integra todos los componentes de un ordenador u otro sistema electrónico.

U-boot. Sistema de arranque que permite empaquetar las instrucciones del kernel para arrancar el sistema operativo del dispositivo. Es un sistema software libre que soporta varios tipos de arquitecturas entre ellas ARM, MIPS y x86. Se encuentra en la mayoría de sistemas empotrados.

Verificación hardware-in-the-loop. Se verifica el funcionamiento del sistema empotrado sobre el hardware real.

Verificación software-in-the-loop. Se verifica el funcionamiento del sistema empotrado sobre una simulación del hardware.

Apéndice C

Manual de Instalación

C.1. Instalación de RTEMS-BeagleaBoard-XM

Para instalar RTEMS, lo primero que se debe realizar es la creación del entorno de desarrollo. Consiste en instalar herramientas (como el compilador, depurador, etc) de la *Board Support Package (BSP)* y el *kernel* de RTEMS. El BSP es la capa del software que contiene especificaciones hardware de drivers. Además, contiene rutinas que permiten a RTEMS funcionar en un entorno de hardware específico. Antes de comenzar, es necesaria la instalación de algunas dependencias.

Lo primero que hacemos es instalar las herramientas necesarias para el desarrollo de RTEMS. Los comandos a ejecutar se pueden ver en el listado C.1

Listado C.1: Herramientas para RTEMS

```
# Instalamos las herramientas necesarias para la compilacion e instalacion de RTEMS
$ sudo apt-get install  g++ gdb git unzip pax bison  flex libpython-dev git
    libncurses5-dev zlib1g-dev
$ sudo apt-get install python-dev
$ sudo apt-get install texinfo
```

Configuración del Source Builder

El *Source Builder* tiene la función de automatizar toda la instalación del entorno de desarrollo necesario para los distintas arquitecturas. Una vez instaladas las herramientas necesarias, se descarga el *Source Builder* y se comprueba que el sistema tiene todos los requisitos necesarios para compilar RTEMS, tal y como se puede ver en el listado C.2.

Listado C.2: Descarga del Source Builder

```
# Clonamos el repositorio de Git del Source Builder
$ cd
$ mkdir -p development/rtems
$ cd development/rtems

$ git clone git://git.rtems.org/rtems-source-builder.git rsb
```

```
# Comprobamos que el source builder ha sido clonado correctamente
$ cd rsb
$ ./source-builder/sb-check
```

Instalación de Build Set

Para cada *target* (i386, Sparc, ARM, etc.) es necesario instalar un conjunto distinto de herramientas de compilación (conocido como *build set*) a través del *Source Builder*. En el listado C.3, se muestra el caso para ARM:

Listado C.3: Build Set para ARM

```
# Descargamos y compilamos todos los archivos binarios de RTEMS para ARM
$ cd rtems
$ ../source-builder/sb-set-builder --prefix=$HOME/development/rtems/5 5/rtems-arm

# Se incluyen los archivos binarios en el PATH.
$ export PATH=$HOME/development/rtems/5/bin:$PATH
```

Compilación

En el listado C.4, se puede observar el proceso de configuración de RTEMS para una arquitectura ARM.

Listado C.4: Configuración de RTEMS para ARM

```
# Clonamos el repositorio de RTEMS y lo configuramos mediante bootstrap:
$ cd
$ cd development/rtems
$ mkdir kernel
$ cd kernel
$ git clone git://git.rtems.org/rtems.git rtems
$ cd rtems
$ ./bootstrap -c && $HOME/development/rtems/rsb/source-builder/sb-bootstrap
```

Para compilar un BSP para una arquitectura específica, sen debe seguir los pasos indicados en el listado C.5.

Listado C.5: Compilacion de BSP beagleboardxm

```
# Se compila RTEMS para el BSP beagleboardxm. El flag --enable-rtems-debug permite
    depurar la aplicacion y el flag -enable-tests=samples genera los test de prueba:
$ cd ..
$ mkdir beagleboardxm
$ cd beagleboardxm
$ $HOME/development/rtems/kernel/rtems/configure --prefix=$HOME/development/rtems/5
     --target=arm-rtems5 --enable-rtemsbsp=beagleboardxm --enable-posix -enable-
    tests=samples --enable-rtems-debug
$ make all
$ make install
```

Para hacer más sencilla esta instalación, se han generado tres scripts:

- `source_builder_RTEMS.sh`: En este script se realiza la descarga del Source Builder, la comprobación de que el sistema tiene todos los requisitos necesarios para compilar RTEMS y la instalación del conjunto de herramientas de compilación.
- `kernel_RTEMS.sh`: En este script se añaden las herramientas compiladas al `PATH`, se descarga el repositorio de RTEMS, se ejecuta el script de configuración y se compila el BSP.
- `instalacion_RTEMS.sh`: Este script realiza las instalación de las herramientas necesarias para el proceso para, a continuación, invocar a los dos script descritos anteriormente.

Como resultado de la instalación de RTEMS para i386, se genera la estructura de directorios descrita al inicio del capítulo. Además, se generan una serie de programas para poder realizar pruebas sin necesidad de generarlas y compilarlas de cero.

C.2. Generar un sistema empotrado con Linux

Como se ha descrito en la sección 4.1, para estudiar la viabilidad del desarrollo de la creación de un sistema empotrado, se ha decidido generar una imagen de disco que ejecuta el conjunto de herramientas BusyBox. El proceso se ha dividido en diversos scripts, donde destacan los comandos del listado C.6.

Listado C.6: Comandos destacados del la creación del sistema empotrado que ejecuta BusyBox

```
1  # Se crea una imagen de disco, donde ${IMAGE} corresponde al nombre de la imagen y
       ${SIZE} al tamano de la misma.
2  $ dd if=/dev/zero of=${IMAGE} count=${SIZE} bs=1048576
3
4  # Se crea la tabla de particiones, una particion y se activa dicha particion.
5  $ parted --script ${IMAGE} mklabel msdos mkpart primary ext4 1 ${SIZE} set 1 boot
       on
6
7  # Se crea un sitema de fichero ext4 en la primera particion.
8  $ mkfs.ext4 /dev/mapper/${LOOPDEV}
9
10 # Se crea el archivo de configuracion de grub
11 $ mkdir -p ${TEMP_DIR}/${GRUB_DIR}
12 $ cat > ${TEMP_DIR}/${GRUB_DIR}/grub.cfg << EOF
13 $ timeout=5
14 $ menuentry 'linux' {
15 $         root=hd0,1
16 $         linux /boot/kernel-3.2.12-gentoo root=/dev/sda3
17 $ }
18 $ EOF
19
20 # Se generan los enlaces a todos los programas de busybox
21 $ for i in $($BUSYBOX --list-full); do
22 $     mkdir -p  ${TEMP_DIR}/$(dirname ${i})
23 $     ln -s /bin/busybox  ${TEMP_DIR}/${i}
24 $ done
25
26 # Se descomprime el contenido de la configuracion etc.
27 $ cd ${TEMP_DIR}
28 $ tar zxvf ../${ETC_FILE}
```

```
$ cd ..

# Se descomprime el fichero que contiene el kernel.
$ cd ${TEMP_DIR}
$ tar zxvf ../${KERNEL}
$ cd ..

# Se convierte la imagen a formato vmdk
$ qemu-img convert -f raw $1 -O vmdk ${1%.*}.vmdk
```

C.3. Instalación U-Boot

Para compilar el sistema de arranque U-Boot para BeagleBoard es necesario ejecutar los comandos que se muestran en el listado C.7.

Listado C.7: Guía para compilar U-Boot

```
# Clonamos el repositorio de U-Boot y lo almacenamos en una carpeta
$ cd
$ sudo apt-get install gcc-arm-linux-gnueabi
$ git clone git://git.denx.de/u-boot.git && cd u-boot

# Configuramos U-Boot para BeagleBoard y lo compilamos
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- make omap3_beagle_defconfig
$ ARCH=arm CROSS_COMPILE=arm-linux-gnueabi- make
```

C.4. Instalación Qemu-Linaro

Para compilar la extensión qemu-linaro se debe instalar el compilador Clang, que se puede ver en el listado 4.6 y ejecutar los comandos del listado C.8.

Listado C.8: Guía para compilar Qemu-linaro

```
# Instalamos las dependencias necesarias
$ sudo apt-get install cmake

# Clonamos el repositorio de qemu-linaro
$ cd
$ git clone -b master --depth=1 http://git.linaro.org/qemu/qemu-linaro.git/
$ cd qemu-linaro/

# Antes de compilar se debe modificar los ficheros qga/commands-posix.c y hw/9pfs/
    virtio-9p.c para incluir la siguiente linea en sus cabeceras: include <sys/
    sysmacros.h>.
$ cmake -j4
```

C.5. cFE

Para poder instalar y compilar la plataforma en un sistema operativo Ubuntu procedemos a ejecutar los siguientes pasos en la terminal:

Para la instalación y compilación de cFE es necesario clonar el repositorio Git de NASA y seguir los pasos del listado C.9.

Listado C.9: Guía para clonar cFE

```
# Clonamos el repositorio de Git
$ git clone https://github.com/nasa/cFE.git

# Obtenemos los submodulos
$ cd cFE
$ git submodule init
$ git submodule update
```

Procedemos a compilar el código del cFE y lo ejecutamos. El procedimiento se puede ver en el listado C.10

Listado C.10: Guía para lo que compilar y ejecutar cFE

```
# Establecemos las variables de enotorno.
$ . ./setvars
$ cd build/cpu1

# Compilamos cFE.
$ make config
$ make

#Ejecutamos cFE.
$ cd exe
$ sudo ./core-linux.bin
```

Bibliografía

[1] NASA, "Big software for smallsats: Adapting cfs to cubesat missions." https://pdfs.semanticscholar.org/45d2/cb1f12e3ab8746c5a8d73094cad3d1a9abc2.pdf. Última visita; 2 de Marzo de 2019.

[2] T. Alcorn, "The sputnik program." https://novaonline.nvcc.edu/eli/evans/his135/events/sputnik57/background.html. Última visita; 29 de Marzo de 2019.

[3] U. de Oviedo, "Sistemas operativos de tiempo real." http://isa.uniovi.es/docencia/TiempoReal/Recursos/temas/sotr.pdf. Última visita; 31 de Marzo de 2019.

[4] P. determinar, "Guía básica de nanosatélites." https://alen.space/es/guia-basica-nanosatelites/. Última visita; 2 de Marzo de 2019.

[5] T. in Virtual, "Diferencias entre i386, x86, x64, amd64 e ia64." https://www.thinkinvirtual.com/2017/03/diferencias-entre-i386-x86-x64-amd64-e.html. Última visita; 30 de Marzo de 2019.

[6] T. Town, "Intel releases its last ever itanium processor." https://www.tweaktown.com/news/57537/intel-releases-last-itanium-processor/index.html. Última visita; 30 de Marzo de 2019.

[7] F. Semiconductor, "Freescale powerpc™ architecture primer." https://www.nxp.com/docs/en/white-paper/POWRPCARCPRMRM.pdf. Consultado; capítulo 2: páginas 11-12.

[8] ARM, "Arm cpu architecture." https://developer.arm.com/architectures/cpu-architecture. Última visita; 30 de Marzo de 2019.

[9] I. SPARC International, "The sparc architecture manual." https://cr.yp.to/2005-590/sparcv9.pdf. Consultado; capítulo 0: páginas 1-2.

[10] ARM, "A-profile architectures." https://developer.arm.com/architectures/cpu-architecture/a-profile. Última visita; 30 de Marzo de 2019.

[11] ARM, "M-profile architectures." https://developer.arm.com/architectures/cpu-architecture/m-profile. Última visita; 30 de Marzo de 2019.

[12] ARM, "Arm cpu architecture." https://developer.arm.com/architectures/cpu-architecture. Última visita; 30 de Marzo de 2019.

[13] BeagleBoard, "Meet the beagles: Open source computing." https://beagleboard.org/. Última visita; 10 de Abril de 2019.

[14] T. Instruments, "Omap3530 and omap3525applicationsprocessors." http://www.ti.com/lit/ds/symlink/omap3530.pdf. Última visita; 10 de Abril de 2019.

[15] BeagleBoard, "Boards." https://beagleboard.org/boards. Última visita; 10 de Abril de 2019.

[16] DigiKey, "Precio de beagleboard original." https://www.digikey.es/product-detail/es/circuitco-electronics-llc/BB-BONE-000/BB-BONE-000-ND/2765688. Última visita; 24 de Abril de 2019.

[17] BeagleBoard, "Beaglebone black." https://beagleboard.org/black. Última visita; 10 de Abril de 2019.

[18] DigiKey, "Precio de beaglebone black." https://www.digikey.es/product-detail/es/ghi-electronics-llc/BBB01-SC-505/BBB01-SC-505-ND/6210999. Última visita; 24 de Abril de 2019.

[19] BeagleBoard, "Beaglebone blue." https://beagleboard.org/blue. Última visita; 10 de Abril de 2019.

[20] DigiKey, "Precio de beaglebone blue." https://www.digikey.es/product-detail/es/ghi-electronics-llc/BBBLE-SC-568/BBBLE-SC-568-ND/7071862. Última visita; 24 de Abril de 2019.

[21] BeagleBoard, "Beagleboard-x15." https://beagleboard.org/x15. Última visita; 10 de Abril de 2019.

[22] Arrow, "Precio de beagleboard x15." https://www.arrow.com/es-mx/products/beagleboardx15/beagleboardorg?utm_source=google&utm_campaign=1662592501&utm_medium=cpc&utm_term=beagleboardx15&utm_currency=EUR&gclid=EAIaIQobChMI4Lvysf_o4QIVGobVCh0YbwMgEAAYAiAAEgLZyfD_BwE&gclsrc=aw.ds. Última visita; 24 de Abril de 2019.

[23] BeagleBoard, "Beagleboard-xm." http://beagleboard.org/beagleboard-xm. Última visita; 13 de Abril de 2019.

[24] DigiKey, "Precio de beagleboard xm." https://www.digikey.es/product-detail/es/precision-technology-inc/BEAGLEBOARD-XM/1777-1000-ND/6834153. Última visita; 24 de Abril de 2019.

[25] UNED, "Sistemas operativos para tiempo real(rtos)." http://www.ieec.uned.es/investigacion/Dipseil/PAC/archivos/Informacion_de_referencia_ISE4_2_2.pdf. Última visita; 31 de Marzo de 2019.

[26] W. River, "Vxworks." https://www.windriver.com/products/product-overviews/2691-VxWorks-Product-Overview.pdf. Última visita; 31 de Marzo de 2019.

[27] Operatingsystem, "Vxworks." https://www.operating-system.org/betriebssystem/_english/bs-vxworks.htm. Última visita; 31 de Marzo de 2019.

[28] FreeRTOS, "Freertos." https://www.freertos.org/. Última visita; 31 de Marzo de 2019.

[29] MarteOS, "Marteos." https://marte.unican.es/. Última visita; 31 de Marzo de 2019.

[30] RTEMS, "Rtems real time operating system (rtos)." https://www.rtems.org/. Última visita; 31 de Marzo de 2019.

[31] NASA, "Cfe." https://opensource.gsfc.nasa.gov/projects/cfe/index.php. Última visita; 31 de Marzo de 2019.

[32] NASA, "Cfs." https://cfs.gsfc.nasa.gov/cFS-OviewBGSlideDeck-ExportControl-Final.pdf. Última visita; 31 de Marzo de 2019.

[33] D. S. Engineering, "Das u-boot – the universal boot loader." http://www.denx.de/wiki/U-Boot/. Última visita; 3 de Mayo de 2019.

[34] Xillybus, "Preparing a uboot image for altera's cyclone v soc fpga." http://xillybus.com/tutorials/u-boot-image-altera-soc. Última visita; 3 de Mayo de 2019.

[35] Frost and Sullivan, "Infoespacial: Frost and sullivan prevén 11.476 lanzamientos de nanosatélites hasta 2030." http://www.infoespacial.com/mundo/2019/01/14/noticia-frost-sullivan-pronostican-11476-lanzamientos-nanosatelites-hasta.html. Última visita; 2 de Marzo de 2019.

[36] U. P. de Cataluña, "Cubecat-1." https://nanosatlab.upc.edu/en/missions-and-projects/3cat-1. Última visita; 2 de Marzo de 2019.

[37] E. Press, "Europa press: Segundo nanosatélite catalán que alcanza la órbita terrestre." https://www.europapress.es/ciencia/misiones-espaciales/noticia-segundo-nanosatelite-catalan-alcanza-orbita-terrestre-20181129133821.html. Última visita; 2 de Marzo de 2019.

[38] U. P. de Cataluña, "Cubecat-2." https://nanosatlab.upc.edu/en/missions-and-projects/3cat-2. Última visita; 2 de Marzo de 2019.

[39] U. P. de Cataluña, "Cubecat-3." https://nanosatlab.upc.edu/en/missions-and-projects/cube-cat-3. Última visita; 2 de Marzo de 2019.

[40] U. P. de Cataluña, "Cubecat-4." https://nanosatlab.upc.edu/en/missions-and-projects/3cat-4. Última visita; 2 de Marzo de 2019.

[41] U. P. de Madrid, "La upm crea el satélite qbito para estudiar la termosfera." http://www.aulamagna.com.es/la-upm-crea-el-satelite-qbito/. Última visita; 18 de Marzo de 2019.

[42] E. O. Portal, "Qbito." https://directory.eoportal.org/web/eoportal/satellite-missions/q/qbito. Última visita; 18 de Marzo de 2019.

[43] Xatcobeo, "Xatcobeo objetives." https://www.xatcobeo.com/cms/index.php?option=com_content&view=article&id=2&Itemid=3. Última visita; 22 de Marzo de 2019.

[44] F. de Vigo, "El xatcobeo finaliza su misión al desintegrarse tras superar los dos años y medio en órbita." https://www.farodevigo.es/gran-vigo/2014/09/03/xatcobeo-finaliza-mision-desintegrarse-superar/1086710.html. Última visita; 22 de Marzo de 2019.

[45] RTEMS, "Manual de rtems." https://docs.rtems.org/branches/master/user/index.html. Última visita; 1 de Mayo de 2019.

[46] Qemu, "Qemu." https://www.qemu.org/. Última visita; 3 de Mayo de 2019.

[47] Linaro, "Qemu-linaro." https://wiki.linaro.org/WorkingGroups/ToolChain/QEMU. Última visita; 3 de Mayo de 2019.

[48] G. de España, "Cuadro nacional de atribución de frecuencias." https://avancedigital.gob.es/espectro/Paginas/cnaf.aspx. Última visita; 25 de Mayo de 2019.

[49] CCSDS, "The consultative committee for space data systems." `https://public.ccsds.org/default.aspx`. Última visita; 25 de Mayo de 2019.

[50] CCSDS, "Areas ccsds." `https://public.ccsds.org/Publications/default.aspx`. Última visita; 25 de Mayo de 2019.

[51] ECSS, "The european cooperation for space standardization." `https://ecss.nl/`. Última visita; 25 de Mayo de 2019.

[52] ESA, "Estándares esa." `http://www.esa.int/TEC/Software_engineering_and_standardisation/TECMKDUXBQE_0.html`. Última visita; 25 de Mayo de 2019.

[53] U. N. O. of Outer Space Affairs, "United nations treaties and principles on space law." `https://web.archive.org/web/20051125013555/http://www.oosa.unvienna.org/SpaceLaw/treaties.html`. Última visita; 22 de Mayo de 2019.

[54] U. N. O. of Outer Space Affairs, "Treaty on principles governing the activities of states in the exploration and use of outer space." `http://www.unoosa.org/oosa/en/ourwork/spacelaw/treaties/travaux-preparatoires/outerspacetreaty.html`. Última visita; 22 de Mayo de 2019.

[55] U. N. O. of Outer Space Affairs, "Convention on international liability for damage caused by space objects." `http://www.unoosa.org/oosa/en/ourwork/spacelaw/treaties/travaux-preparatoires/liability-convention.html`. Última visita; 22 de Mayo de 2019.

[56] U. N. O. of Outer Space Affairs, "Convention on registration of objects launched into outer space." `http://www.unoosa.org/oosa/en/ourwork/spacelaw/treaties/registration-convention.html`. Última visita; 22 de Mayo de 2019.

[57] T. Guardian, "Nasa budgets: Us spending on space travel since 1958." `https://www.theguardian.com/news/datablog/2010/feb/01/nasa-budgets-us-spending-space-travel`. Última visita; 31 de Mayo de 2019.